# Tensor Approximation Methods for Stochastic Problems

Von der Carl-Friedrich-Gauß-Fakultät
der Technischen Universität Carolo-Wilhelmina zu
Braunschweig

zur Erlangung des Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigte Dissertation

von
Elmar Klaus Zander
geboren am 21. April 1970
in Bottrop

Eingereicht am: 12. Januar 2012
Disputation am: 22. Mai 2012
1. Referent: Prof. Dr. Herrmann G. Matthies
2. Referent: Prof. Dr. Anthony Nouy

Druckdatum: 16. März 2017

## Abstract

Spectral stochastic methods have gained wide acceptance as a tool for efficient modelling of uncertain stochastic systems. The advantage of those methods is that they provide not only statistics, but give a direct representation of the measure of the solution as a so-called surrogate model, which can be used for very fast sampling.

Especially attractive for elliptic stochastic partial differential equations (SPDEs) is the stochastic Galerkin method, since it preserves essential properties of the differential operator. One drawback of the method is, however, that it requires huge amounts of memory, as the solution is represented in a tensor product space of spatial and stochastic basis functions. Different approaches have been investigated to reduce the memory requirements, for example, model reduction techniques using subspace iterations to reduce the approximation space or methods of approximating the solution from successive rank-1 updates.

In the present thesis best approximations to the solutions of linear elliptic SPDEs are constructed in low-rank tensor representations. By using tensor formats for all random quantities, the best subsets for representing the solution are computed "on the fly" during the entire process of solving the SPDE. As those representations require additional approximations during the solution process it is essential to control the convergence of the solution. Furthermore, special issues with preconditioning of the discrete system and stagnation of the iterative methods need adequate treatment. Since one goal of this work was practical usability, special emphasis has been given to implementation techniques and their description in the necessary detail.

## Kurzfassung

Spektrale stochastische Methoden haben sich als effizientes Werkzeug zur Modellierung von Systemen mit Unsicherheiten etabliert. Der Vorteil dieser Methoden ist, dass sie nicht nur Statistiken liefern, sondern auch eine direkte Darstellung der Lösung als sogenanntes Surrogatmodell.

Besonders attraktiv für elliptische stochastische partielle Differentialgleichungen (SPDGln) ist das stochastische Galerkin Verfahren, da in diesem wesentliche Eigenschaften des Differentialoperators erhalten bleiben. Ein Nachteil der Methode ist jedoch, dass enorme Mengen an Speicherplatz benötigt werden, da die Lösung in einem Tensorprodukt der räumlichen und stochastischen Ansatzräume liegt. Bisher wurden verschiedene Ansätze erprobt, um diese Anforderung zu verringern. Hierzu zählen Modellreduktionstechniken, Unterraumiterationen, um den Lösungsraum auf einen beherrschbaren Unterraum einzuschränken, oder Methoden, welche die Lösung schrittweise aus Rang-1 Produkten aufzubauen.

In der vorliegenden Arbeit werden Bestapproximationen der Lösungen linearer SPDGln als Niedrig-Rang-Darstellungen gesucht. Dies wird dadurch erreicht, dass Tensordarstellungen sowohl für die Eingangsdaten als auch für die Lösung verwendet und während des ganzen iterativen Lösungsprozesses beibehalten werden. Da diese Darstellungen weitere Näherungen während des Lösungsprozesses erfordern, ist es wesentlich die Konvergenz der Lösung genau zu überwachen. Ferner müssen Besonderheiten der Präkonditionierung der diskreten Systeme und der Stagnation der iterativen Verfahren beachtet werden. Mit dem Ziel der praktischen Anwendbarkeit als einem wesentlichen Bestandteil dieser Arbeit wurde großer Wert auf eine detaillierte Beschreibung der Implementierungstechniken gelegt.

## Acknowledgements

First and foremost, I would like to express my gratitude to my advisor Prof. Dr. Hermann G. Matthies, head of the Institute of Scientific Computing at the Technische Universität Braunschweig, for introducing me to the subject of stochastic partial differential equations, for the inspiration underlying this research work, and for the many constructive discussions. I would also like to thank Prof. Dr. Anthony Nouy from the Université de Nantes, France, for agreeing to referee my thesis and his constructive remarks.

Further thanks go to Dr. Alexander Litvinenko and Dr. Martin Eigel for revising this thesis and Cosima Meyer, M.A., for proofreading. My gratefulness goes also to all current and former colleagues at the Institute of Scientific Computing for the constructive and cooperative working environment.

I am also grateful to people in the group of Prof. Dr. Wolfgang Hackbusch at the Max-Planck-Institute in Leipzig, especially Prof. Dr. Lars Grasedyck and Dr. Mike Espig from whom I could learn much about tensors and low-rank formats.

Last, but not least, my special gratitude goes to my wife, Heike Zander, for her patient support and constant encouragement.

# Contents

# Chapter 1

# Introduction

In the past decades great progress has been achieved in the numerical sciences with regard to the speed and accuracy of numerical simulations of technical or natural systems. Nowadays, many problems can be solved to very small prescribed numerical errors. As long as the mathematical model accurately describes the essential aspects of the physical system under investigation, methods have been developed to effectively control the error in order to give good estimates on the reliability of the computed solutions.

This, however, is generally only an estimate of the *numerical error* involved. Problems usually include real world data as inputs, which are themselves contaminated by measurement errors and/or based on estimated or interpolated data due to lack of real data. The effect those uncertainties in the input data have on the computed solutions is generally not quantified by classical numerical methods.

In the currently very active research area of uncertainty quantification (UQ) this problem is addressed. In one branch of UQ the uncertainties in the investigated systems are modelled as stochastic entities, and statistics of the system solution are generated from the statistics of the input data. The two main strands here are first the sampling based approaches, which solve the systems for many realisations of input data that are statistically consistent with the measured data. The second approach, that is followed in this thesis, regards the input data and the solution as random variables or random fields which can be directly described by a spectral representation.

A drawback of the first approach is that often a huge number of deterministic realisations of the system have to be solved in order to get reliable statistics. This can be overcome by the second approach, however, at the cost of greatly increasing the dimensionality of the linear systems that need to be solved. This work is concerned with the reduction of this cost by employing so-called tensor product representations for the involved quantities throughout the solution process.

## 1.1 Summary

This chapter gives a short overview of the computational problems that arise when solving linear systems stemming from the stochastic Galerkin discretisation of linear elliptic SPDEs. Different recent approaches for the reduction of the size and the efficient solution of those systems are summarised and contrasted with the method developed in this work.

Chapter 2 discusses basics of the algebraical and topological tensor products, as they form the basis on which stochastic PDEs and their discretisation can be understood. The numerical treatment of tensors is deferred to Chapter 4, until after the discretisation of SPDEs has been discussed.

Chapter 3 presents the discretisation of stochastic PDEs arising from problems in uncertainty quantification using stochastic Galerkin methods. It is demonstrated that the resulting linear systems have tensor product form, whose memory and time efficient solution depends strongly on the representation of those tensors.

In Chapter 4 the numerical treatment of tensors is discussed as far as it is relevant for the solution process of discretised SPDEs. For higher order tensors this is also referred to as multilinear algebra. The arithmetic on tensor products is described in a general way first and then made concrete for the different representations for second order tensors. Formats for higher order tensors are discussed briefly at the end of the chapter.

Chapter 5 analyses the effect of the perturbations induced by

the use of tensor representations and the necessary truncations on general iterative processes that build the framework for every linear solver. A priori and a posteriori error estimates are derived and applied to concrete iterative methods in the following chapter. Furthermore, strategies to detect stagnation of the iterations and to dynamically adapt the truncation parameter are presented.

Chapter 6 discusses the use of tensor representations in concrete linear solvers for the linear systems arising in Chapter 3. First, simple iterative methods similar to the Jacobi or Richardson method are treated, as the theory from the previous chapter applies here directly. Then the conjugate gradient method is also treated, since this method usually shows faster convergence than the stationary methods.

Chapter 7 examines issues with preconditioning that arise particularly in the context of tensor methods. The first section deals with preconditioning of tensor operators, while the second deals with preconditioning strategies that are advantageous, when the solution is represented in a low-rank tensor format.

In Chapter 8 numerical results for the application of the tensor product methods of the previous chapter in the context of SPDEs are shown and discussed. The focus lies on convergence, as well as on robustness. Finally, the tensor methods are compared to standard methods in terms of speed and memory efficiency.

## 1.2 Problem statement

In order to show the numerical and computational issues involved in solving stochastic partial differential equations with the stochastic Galerkin method, we will begin with a brief sketch of how to solve such an equation numerically; for details see e.g. [5, 28, 61, 62]. Consider the linear stochastic PDE

$$\mathcal{L}(\omega)(u(x,\omega)) = f(x,\omega) \qquad x \in \mathcal{D}, \tag{1.1}$$

where $\mathcal{L}(\omega)$ is a linear differential operator depending on a stochastic parameter $\omega \in \Omega$. The equation shall be fulfilled $\mathbb{P}$-almost

surely on the space of random events $\Omega$, where $\mathbb{P}$ is some probability measure defined on the $\sigma$-algebra $\mathcal{F} \subset 2^{\Omega}$.

In the context of spectral stochastic methods, a variational formulation of Eq. (1.1) is employed for which, depending on regularity conditions on $\mathcal{L}$, well-posedness of the problem can be ensured. The solution $u$ is required to be in a variational space $\mathcal{X} \otimes \mathcal{S}$. Common choices are $\mathcal{X} = H^1_E(\mathcal{D})$ for second order linear differential operators defined on some domain $\mathcal{D}$ and $\mathcal{S} = L_2(\Omega, \mathcal{F}, \mathbb{P})$ for random fields with finite variance.

Given finite dimensional subspaces $\mathcal{X}_N \subset \mathcal{X}$ and $\mathcal{S}_M \subset \mathcal{S}$ the Galerkin principle can be applied to Eq. (1.1) yielding a best approximation $u_{NM}$ for $u$ in the space $\mathcal{X}_N \otimes \mathcal{S}_M$. Specifying bases for $\mathcal{X}_N$ and $\mathcal{S}_M$ leads to a discrete linear system

$$\mathbf{K}\mathbf{u} = \mathbf{f} \tag{1.2}$$

where $\mathbf{u} \in \mathbb{R}^N \otimes \mathbb{R}^M$ contains the coefficients of $u_{NM}$ with respect to the tensor product basis of $\mathcal{X}_N \otimes \mathcal{S}_M$. The discrete operator $\mathbf{K}$ has the following tensor product structure

$$\mathbf{K} = \sum_{k=0}^{L} \boldsymbol{K}_k \otimes \boldsymbol{\Delta}_k, \tag{1.3}$$

with $\boldsymbol{K}_k \in \mathbb{R}^{N \times N}$ and $\boldsymbol{\Delta}_k \in \mathbb{R}^{M \times M}$. Typical (but not necessary) choices are finite elements bases for $\mathcal{X}_N$ and the polynomial chaos (PC) for $\mathcal{S}_M$, respectively. Commonly the discrete solution $\mathbf{u}$ is represented as element of $\mathbb{R}^{NM}$, i.e. a long vector or, equivalently, as a matrix in $\mathbb{R}^{N \times M}$. The discrete system Eq. (1.2) can then be solved e.g. by standard block matrix or Krylov subspace methods.

The problem that arises here is that (a) to store the full solution $NM$ units of memory are necessary, and (b) the application of the operator $\mathbf{K}$ can become very costly. This can quickly become infeasible, since $N$ and $M$ are in general already large numbers themselves, as $N$ comes from the finite element discretisation and $M$ grows quickly with the number of random variables and the polynomial degree. Different approaches have been developed to alleviate this problem, which shall be briefly described in the

following section.

## 1.3 State of the art

The methods described in the literature to solve the aforementioned problem can be roughly divided into the following categories, where problems originating from other fields than uncertainty quantification, but leading to the same type of equations, have also been included:

**Model order reduction approaches (MOR):** The aim is to find subspaces of $\mathcal{X}_N$ and $\mathcal{S}_M$ with good approximation properties into which the equation is subsequently projected and solved.

The probably most prominent approach of this type is the proper generalised decomposition (PGD), formerly also called generalised spectral decomposition (GSD), introduced by Nouy in [66]. The basic idea for the computation of the subspaces is the following: from a given fixed subspace of $\mathcal{X}_N$ compute a best approximating subspace of $\mathcal{S}_M$ in the least squares sense. Then keep this subspace fixed and find a best subspace in $\mathcal{X}_N$, and alternate. This can be seen as a generalised eigenproblem, that can be solved using subspace or Arnoldi iterations [68]. The approach has been applied to a variety of problems, e.g. to nonlinear [17], time-dependent [67] and high-dimensional stochastic problems [69].

**Approximation by successive rank-1 updates:** The solution is computed iteratively using successive rank-1 updates that are in general computed in a greedy fashion by some form of optimisation without computing the relevant subspaces beforehand.

In an approach by Krosche and Niekamp [54] the solution is represented as a sum of elementary tensors of second order, which are computed successively as rank-1 updates of the already computed low-rank solution. The next rank-1

update is computed by approximately solving a minimisation problem for the variational form associated with the given differential operator. Since this greedy approach does not necessarily lead to a best rank-$k$ approximation after $k > 1$ steps, an extension has been proposed, which explicitly minimises the energy error.

Falco and Nouy [23, 24] propose to use progressive PGD to construct separated representations of the solution, i.e. as a sum of elementary tensors. Since this is a greedy algorithm, which is not guaranteed to find the optimal representation, update steps are proposed to recover the optimality that would be achieved if the optimisation were performed simultaneously. Convergence is proved for certain high-dimensional Laplace problems in Banach spaces [24] and in the more general setting of linear elliptic variational problems in tensor Hilbert spaces [23].

**Low-rank/tensor representation of the solution:** In this approach a low-rank representation is kept throughout the solution process. This is the approach the current thesis follows.

Ballani and Grasedyck developed a low-rank solver for linear systems arising in quantum mechanics based on a GMRES projection method [8]. The system operator is linear and has a low-rank high-order tensor structure, as it originates from the discretisation of the Hamilton operator. The tensors themselves are represented in the hierarchical Tucker format [37], which exhibits some advantages in truncation efficiency for higher order tensors compared to e.g. the higher order SVD [55]. Since fixed-rank truncations are used, no convergence results can be given, however.

A work that is similar in spirit to the present one has recently been published by Khoromskij and Schwab [49]. The solution to an elliptic stochastic PDE is computed by using the canonical tensor format in preconditioned simple iterations. The algorithm is reported to be efficient for

higher order tensors, but not disclosed in the article. The proposed method works with higher order tensor, however, under simplified assumptions on the stochastic operator (facilitating higher order representations) and very low rank of the stochastic solution as the problem treated had a one-dimensional spatial domain.

**Sparse tensor product spaces:** In this approach by Bieri [12, 13] the spaces $\mathcal{S}_M$ and $\mathcal{X}_N$ are hierarchically decomposed into so-called *detail spaces*. Instead of searching the solution in the full tensor product space $\mathcal{X}_N \otimes \mathcal{S}_M$, the solution is sought in a direct sum of tensor products of the detail spaces.

**Reducing the size of the stochastic basis:** This approach is essentially independent of the ones mentioned before as here the stochastic basis itself is modified.

In an approach by Doostan et al. [20] the system is first solved with a coarse spatial approximation, but with the full stochastic basis. The coarse solution obtained is then used to determine and select the most important stochastic basis functions. After that the system is solved using the fine discretisation and the reduced subset of stochastic basis functions determined in the previous step.

El Moselhy and Marzouk [21] developed an algorithm that starts with a small stochastic basis and then adaptively selects more basis functions from a pool of "proposed" basis functions. Only those are taken that show a significant contribution in the residual, when measured with respect to an even larger basis (which can be truly huge). The set of proposed functions is then adaptively changed. One drawback of the proposed algorithm is, that currently only log-normal random fields can be used as input for the algorithm, since it relies on special properties of the log-normal distribution in order to work efficiently.

## 1.4 Objectives

The objective of this thesis was to devise, analyse, implement and test solvers for systems that arise from the discretisation of linear elliptic SPDEs, yielding a low-rank solution that is kept throughout the solution process. The developed solvers should be able to operate on fairly general random field models and not presuppose any particular form of the operator or special properties of the input random fields.

The implementation of the solvers should work efficiently, so that reasonably large test cases could be computed. A driving force for this was also that certain complications only show up for sufficiently large systems or when the ranks of the tensor approximations stop being really "low". Therefore, algorithms for efficiently steering the rank during the iterations (dynamic truncation) and other optimisations (orthogonalisation, preconditioning strategies) were needed. With the developed solvers systems of up to $10^8$ unknowns could be solved on a standard PC workstation.

As necessary theoretical foundation estimates on the convergence of general iterative processes under perturbations had to be developed, as well as estimates for the combined perturbations per iteration step induced by the truncations of the low-rank formats. From there a posteriori error estimates could be derived, which proved to be of practical value in the solvers. Further, criteria for stagnation of the convergence needed to be developed, which also proved to be reliable in practice.

## 1.5 Notes on the implementation

Development of scientific software is a time consuming task. Furthermore, in the process of making an algorithm explicit by coding it in a concrete programming environment, many decisions have to be made that influence the final results the numerical code will compute. Therefore, in the author's view, the description of a numerical method is only then complete, when there is at least

one freely available, prototypical implementation, which shows all those design decisions. Only this allows other interested parties to fully assess the efficiency, robustness and generality of the proposed method.

All methods presented in this work have been implemented in a software library called sglib (shorthand for stochastic Galerkin library) designed to work under Matlab® R2008 and later versions. This library, including all sample codes needed to generate the tables and figures in this work, can be downloaded freely from `https://github.com/ezander/sglib`. sglib is distributed under the free software license GPL version 3.0 [26]. For performing FEM calculations the "PDE Toolbox" version 1.0.15 was used; the library sglib, however, does not depend on this toolbox, and other finite element codes can and have been used as well.

Since software is never truly finished and always in a state of change and improvement, it is necessary for the reproducibility of the results to specify the exact state of the software that was used. The version control system *git* [39], which is used here, uses hash values to uniquely identify the system state (i.e. the *commit*). The results presented in this thesis can be reproduced (except possibly for runtime measurements) by using the sglib version with the following commit hash:

2dee157fa305551268ddcae8e25dfe4b03645b89.

The scripts that generate the tables and figures are generally referred to in the respective captions by "(Script: `<scriptname>`)", where the Matlab® script file `<scriptname>.m` can be found under the path `sglib/thesis/` in sglib. Scripts that are referred to in the text can be found under the path `sglib/`. As a convenience for readers viewing the PDF version of this document, left-clicking the script name will open a browser window, in which the script in the version that was used when writing this thesis is loaded from the project host at `www.github.com`.

# Chapter 2

# Tensors and Tensor Products

*This chapter discusses algebraic and topological properties of tensor products. The exposition starts with the algebraic, coordinate-free definition of the tensor product of vector spaces. This approach facilitates many proofs which are much more tedious, when the classical index-based definition is employed. In the following section extensions to the tensor product for Hilbert spaces are discussed.*

*The treatment is focused on the numerical applications later in this work. For a more thorough exposition the reader may refer to e.g. [78] for the algebraic tensor product and to [80] for tensor products of Banach spaces.*

## 2.1 The algebraic tensor product

Suppose we have vectors spaces $\mathcal{U}$ and $\mathcal{V}$ given and further a vector space $\mathcal{T}$ and a mapping $\tau : \mathcal{U} \times \mathcal{V} \to \mathcal{T}$. If every bilinear map $b : \mathcal{U} \times \mathcal{V} \to \mathcal{W}$, where $\mathcal{W}$ is some vector space, can be factored uniquely as $b = \tilde{b} \circ \tau$, with $\tilde{b} : \mathcal{T} \to \mathcal{W}$, as depicted in Fig. 2.1 then $\mathcal{T}$ is called the (algebraic) tensor product of $\mathcal{U}$ and $\mathcal{V}$.

The existence of such a unique linear map $\tilde{b}$ with the given properties is often called the *universal property of the tensor product*. By standard arguments from category theory, it can be shown that this defines a unique object (see e.g. [78]), which is henceforth written as $\mathcal{T} = \mathcal{U} \otimes \mathcal{V}$ and the injection $\tau = (u, v) \mapsto u \otimes v$. It is interesting to note here that uniqueness in this algebraic context always means "up to isomorphism", which usually means

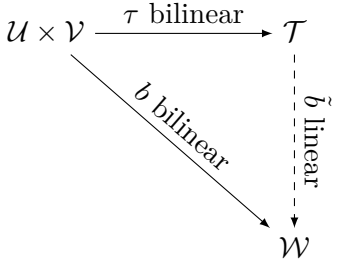



**Figure 2.1:** The tensor product factors the bilinear map $b$ into the universal bilinear map $\tau$ and the unique linear map $\tilde{b}$, i.e. $b = \tilde{b} \circ \tau$

some choice in numerical representation. Exactly this freedom of representation shall be later exploited in this thesis.

From the definition of the tensor product follows its uniqueness, but not the existence of a pair $(\mathcal{T}, \tau)$ that fulfil the conditions. However, there are constructions that work for general vector spaces. The first one relies on the fact that every vector space has a (Hamel) basis, say $e_i$ and $f_j$. Then $\mathcal{T}$ can be constructed as the free vector space over $e_i \otimes f_j$ (which is just a formal way to denote the basis elements), and $\tau$ can be defined for the basis vectors by $\tau(e_i, f_j) = e_i \otimes f_j$ and (uniquely) extended by linearity.

The second construction follows from the fact that the tensor product *uniquely* factors any bilinear form. Since we have $b(\alpha u, v) = b(u, \alpha v) = \alpha b(u, v)$ for any bilinear form $b$ and scalar $\alpha$ by definition of bilinearity, it must also hold that $(\alpha u) \otimes v = u \otimes (\alpha v) = \alpha u \otimes v$ for the tensor product to be unique. Further, from $b(u_1 + u_2, v) = b(u_1, v) + b(u_2, v)$ it can be derived that $u_1 \otimes v + u_2 \otimes v = (u_1 + u_2) \otimes v$ must hold, with an equivalent relation in the second argument. The idea is now to define the tensor product space $T$ by first generating a larger vector space that contains all formal sums of the form $\sum_{i=1}^{n} \alpha_i (u_i, v_i)$ and then factor through the equivalence classes of tensors implied by the action of the bilinear forms on them.

Let $\mathcal{F}_{\mathcal{U} \times \mathcal{V}}$ denote the free vector space over the real numbers

with basis $\mathcal{U} \times \mathcal{V}$ in the sense of sets[1], i.e. the space of all finite formal sums $\sum_i \alpha_i(u_i, v_i)$. Trivially, this space is large enough to factor any bilinear form, however, it is too large, since the above equalities do not hold, and thus the factorisation would not be unique. The equivalence relations derived from the discriminating power of bilinear maps

$$(au, v) \sim (u, av) \sim a(u, v) \tag{2.1}$$

$$(u_1 + u_2, v) \sim (u_1, v) + (u_2, v) \tag{2.2}$$

$$(u, v_1 + v_2) \sim (u, v_1) + (u, v_2) \tag{2.3}$$

define a subspace $S$ of $\mathcal{F}_{\mathcal{U}\times\mathcal{V}}$. Now the quotient space

$$\mathcal{T} = \mathcal{F}_{\mathcal{U}\times\mathcal{V}}/S$$

together with the map $\tau : (u, v) \in \mathcal{U} \times \mathcal{V} \mapsto u \otimes v = (u, v) + S \ni \mathcal{T}$ defines a tensor product.

Both types of constructions can be interpreted numerically and lead to different numerical representations of tensors. However, as both constructions define a tensor product there must be an isometric isomorphism between them. Explicit constructions of such isomorphisms between tensor products of real Cartesian vector spaces are dealt with in the section on numerical representations and arithmetic (see Chapter 4). For higher order tensors there are more sophisticated representations, which do not directly follow from the basic constructions. These will be briefly surveyed in Section 4.2.

The definition of higher order tensors is analogous to the definition of tensor products of two vector spaces by replacing the notion of bilinearity with that of multilinearity. That means, the tensor product, of say three, vector spaces $\mathcal{U}$, $\mathcal{V}$ and $\mathcal{W}$ uniquely factors any trilinear map on $\mathcal{U}\times\mathcal{V}\times\mathcal{W}$. However, higher order tensor products can also be composed of lower order tensor products, since it can be shown that those constructions are isomorphic,

[1] That means that a sum $(u_1, v_1) + (u_2, v_2)$ is left as a formal sum, since $\mathcal{U} \times \mathcal{V}$ is regarded as a mere set without linear structure, and cannot be contracted to $(u_1 + u_2, v_1 + v_2)$

e.g. there is an isomorphism between $\mathcal{U} \otimes \mathcal{V} \otimes \mathcal{W}$ and $(\mathcal{U} \otimes \mathcal{V}) \otimes \mathcal{W}$, i.e.

$$\mathcal{U} \otimes \mathcal{V} \otimes \mathcal{W} \simeq (\mathcal{U} \otimes \mathcal{V}) \otimes \mathcal{W}.$$

The tensor product is thus associative and the parenthesis may be dropped. However, as before, this isomorphism may imply different numerical representations. The number of tensorised vector spaces is called the *order* of the tensor product.

Elements of a tensor space $\mathcal{U} \otimes \mathcal{V}$ are called tensors, and for the image of the map $\tau : \mathcal{U} \times \mathcal{V} \to \mathcal{U} \otimes \mathcal{V}$ often the same symbol is used, such that the element $\tau(u, v)$ is simply denoted as $u \otimes v$. However, not every tensor in $\mathcal{U} \otimes \mathcal{V}$ has this simple form—those that do are called *elementary*, or *decomposable* tensors [78].

However, every tensor $\mathbf{x} \in \mathcal{U} \otimes \mathcal{V}$ allows a representation as a finite sum of elementary tensors, i.e.

$$\mathbf{x} = \sum_{i=1}^{n} u_i \otimes v_i.$$

The smallest such $n$ is called the *rank* of the tensor. This definition of rank easily generalises to higher order tensor products. Note that for the Hilbert space tensor product (see Section 2.2) the rank of a tensor need not be finite.

## 2.2 Tensor products of Hilbert spaces

For Hilbert spaces there is not so much choice as in the case of Banach spaces, if the tensor product of the spaces should be considered a Hilbert space itself again. The reason is that the only natural inner product on $\mathcal{H} \otimes \mathcal{K}$, where $\mathcal{H}$ and $\mathcal{K}$ are Hilbert spaces, is given for elementary tensors by

$$\langle u \otimes v | x \otimes y \rangle_{\mathcal{H} \otimes \mathcal{K}} = \langle u | x \rangle_{\mathcal{H}} \langle v | y \rangle_{\mathcal{K}}. \tag{2.4}$$

and can be extended to general tensors by bilinearity. For general tensors $w = \sum_{i=1}^{m} u_i \otimes v_i$ and $z = \sum_{i=1}^{m} x_i \otimes y_i$ the inner product $\langle w | z \rangle_{\mathcal{H} \otimes \mathcal{K}}$ is independent of the decompositions. The completion

of the algebraic tensor product in the topology induced by this inner product is called the Hilbert space tensor product.

**Remark 2.1.** *Note that a tensor in a tensor product of Hilbert spaces may have infinite rank, if at least one of the spaces is infinite dimensional. This is in contrast to the algebraic tensor product, in which the rank of a tensor is* always *finite.*

**Remark 2.2.** *In analogy to the algebraic case, if $\mathcal{H}$ and $\mathcal{K}$ have orthonormal bases $\{e_i\}_{i\in I}$ and $\{f_j\}_{j\in J}$ then $\mathcal{H}\otimes\mathcal{K}$ has orthonormal basis $\{e_i\otimes f_j\}_{(i,j)\in(I\times J)}$. Therefore, the Hilbert space dimension of $\mathcal{H}\otimes\mathcal{K}$ is the product of the dimension of $\mathcal{H}$ and $\mathcal{K}$.*

**Remark 2.3.** *In contrast to the Banach space tensor product the higher order Hilbert space tensor products are again associative like the algebraic tensor product, i.e.*

$$(\mathcal{H}_1\otimes\mathcal{H}_2)\otimes\mathcal{H}_3 \simeq \mathcal{H}_1\otimes(\mathcal{H}_2\otimes\mathcal{H}_3) \simeq \mathcal{H}_1\otimes\mathcal{H}_2\otimes\mathcal{H}_3 \qquad (2.5)$$

The following interesting and for stochastic PDEs important relations can be derived from this. First, for $L_2$ spaces on product domains $X\times Y$ it can be shown that $L_2(X\times Y)$ is isomorphic to $L_2(X)\otimes L_2(Y)$. Taking some $f\in L_2(X)$ and $g\in L_2(Y)$ we can define an $h\in L_2(X\times Y)$ by $h(x,y)=f(x)g(y)$. This defines a linear map from $L_2(X)\times L_2(Y)\to L_2(X\times Y)$, which has dense image if $L_2(X)$ and $L_2(Y)$ are separable. By taking the completion of the algebraic tensor product with respect to the Hilbert space topology this results in the relation

$$L_2(X)\otimes L_2(Y)\simeq L_2(X\times Y).$$

A second relation involves Hilbert spaces of functions with values in other Hilbert spaces. Let e.g. $f\in L_2(X)$ and $g\in\mathcal{H}$, where $\mathcal{H}$ is some separable Hilbert space. Then we can map $(f,g)$ to $x\mapsto f(x)g$ which is in $L_2(X;\mathcal{H})$, the space of square integrable functions with values in $\mathcal{H}$. The same denseness arguments as before can now be made to conclude that this defines an isomorphism between $L_2(X)\otimes\mathcal{H}$ and $L_2(X;\mathcal{H})$. Furthermore,

this can easily be extended to the Sobolev spaces $H^k$, namely $H^k(X;\mathcal{H}) \simeq H^k(X) \otimes \mathcal{H}$.

## 2.3 Tensor products of operators

Since linear operators on vector spaces form a vector space themselves, a tensor product of operators (or operator vector spaces) can also be constructed. Let $\boldsymbol{A} : \mathcal{U} \to \mathcal{U}'$ and $\boldsymbol{B} : \mathcal{V} \to \mathcal{V}'$ be two linear operators, i.e. $\boldsymbol{A} \in \mathcal{L}(\mathcal{U},\mathcal{U}')$ and $\boldsymbol{B} \in \mathcal{L}(\mathcal{V},\mathcal{V}')$. Then the following theorem holds.

**Theorem 2.4.** *The linear transformation*

$$\lambda : \mathcal{L}(\mathcal{U},\mathcal{U}') \otimes \mathcal{L}(\mathcal{V},\mathcal{V}') \to \mathcal{L}(\mathcal{U} \otimes \mathcal{V},\mathcal{U}' \otimes \mathcal{V}')$$

*defined by*

$$\lambda = (\boldsymbol{A} \otimes \boldsymbol{B}) \mapsto ((\boldsymbol{u} \otimes \boldsymbol{v}) \mapsto \boldsymbol{A}(\boldsymbol{u}) \otimes \boldsymbol{B}(\boldsymbol{v}))$$

*is an embedding. It is further an isomorphism if the vector spaces are finite dimensional. The tensor product $\boldsymbol{A} \otimes \boldsymbol{B}$ of linear operators is thus a linear operator on tensor products.*

*Proof.* For a proof see e.g. [78]. □

In the case that the spaces involved are Hilbert spaces and the operators are Hilbert-Schmidt the embedding $\lambda$ also becomes an isomorphism on the completed space. Then the spaces $\mathcal{L}(\mathcal{U},\mathcal{U}') \otimes \mathcal{L}(\mathcal{V},\mathcal{V}')$ and $\mathcal{L}(\mathcal{U} \otimes \mathcal{V},\mathcal{U}' \otimes \mathcal{V}')$ can be identified, and it is permissible—with a slight abuse of notation—to omit the isomorphism $\lambda$ and directly write

$$(\boldsymbol{A} \otimes \boldsymbol{B})(\boldsymbol{u} \otimes \boldsymbol{v}) = \boldsymbol{A}(\boldsymbol{u}) \otimes \boldsymbol{B}(\boldsymbol{v}),$$

instead of the more cumbersome $\lambda(\boldsymbol{A} \otimes \boldsymbol{B})(\boldsymbol{u} \otimes \boldsymbol{v})$.

# Chapter 3

# Stochastic partial differential equations

*This chapter describes the discretisation of elliptic stochastic PDEs and associated issues. As most of this has already been treated thoroughly in the literature (see e.g. [28, 45, 56, 61, 91]), the exposition is kept brief. The variational formulation for the deterministic equation is derived first, and then extended to the stochastic variational formulation. Restriction to finite dimensional subspaces and application of Galerkin's principle lead to the discrete equations. As tensor methods are central to this thesis, the discrete equations are directly derived in tensor structure.*

*Though the form of the discrete equations is independent of the concrete choice of bases, the particular choice for the discretisation of the stochastic space taken in this thesis, the polynomial chaos expansion, needs to be discussed. Further, the representation of random fields in a finite number of random variables by using the Karhunen-Loève expansion (KLE) will be addressed, and the relation of the KLE to the singular value decomposition (SVD) and tensor representations. The chapter concludes with a method that is used for the synthesis of random fields, needed for the specification of the input uncertainties.*

## 3.1 Variational formulation

Consider first the deterministic boundary value problem

$$-\nabla \cdot (\kappa(x)\nabla u(x)) = f(x), \qquad x \in \mathcal{D},$$

$$\begin{aligned} u(x) &= g(x) && x \in \Gamma_D, \\ \kappa(x)\nabla u(x) \cdot n &= h(x) && x \in \Gamma_N, \end{aligned} \tag{3.1}$$

where $\mathcal{D} \subset \mathbb{R}^d$ is a bounded and sufficiently regular domain (e.g. polygonal or Lipschitz), $\Gamma_D$ is the Dirichlet boundary, and $\Gamma_N$ is the Neumann boundary. In the following, homogeneous boundary conditions, i.e. $g(x) = 0$ and $h(x) = 0$, have been assumed for simplicity of the exposition. However, the treatment of inhomogeneous boundary conditions poses no theoretical difficulty and has been handled in many books on finite element methods (see e.g. [15, 31, 92]).

Let $\kappa \in L_\infty$ with $\kappa(x) \geq \kappa_{\min} > 0$ almost everywhere in $\mathcal{D}$ and $\mathcal{X} = H_E^1(\mathcal{D})$ the Sobolev space of functions with square integrable weak derivatives in $\mathcal{D}$, vanishing on $\Gamma_D$. Then

$$a(u, v) = \int_{\mathcal{D}} \nabla u(x) \cdot \kappa(x) \nabla v(x) \, \mathrm{d}x \tag{3.2}$$

defines a continuous and coercive bilinear form on $\mathcal{X} \times \mathcal{X}$, and thus the variational form of Eq. (3.1), namely

$$a(u, v) = b(v), \qquad v \in \mathcal{X} \tag{3.3}$$

has a unique solution in $\mathcal{X}$, where $b(v) = \int_{\mathcal{D}} f(x)v(x) \, \mathrm{d}x$ is a continuous linear form on $\mathcal{X}$. Existence and uniqueness of the solution follow from the Lax-Milgram lemma (see e.g. [15]).

The variational form Eq. (3.3) is also the starting point for Galerkin's method to find approximations to $u$ in a finite dimensional subspace of $\mathcal{X}$. Let $\mathcal{X}_N$ be an $N$ dimensional subspace of $\mathcal{X}$, then the bilinear form in Eq. (3.2) is also continuous and coercive when restricted to $\mathcal{X}_N \times \mathcal{X}_N$. Thus

$$a(u_N, v_N) = b(v_N), \qquad v_N \in \mathcal{X}_{\mathcal{N}} \tag{3.4}$$

has a unique solution $u_N \in \mathcal{X}_N$.

In standard finite element texts the approximate solution is usually denoted by $u_h$, where $h$ is a mesh parameter (generally the largest edge length of the mesh) and convergence of $u_h \to u$ is

shown for non-degenerate families of meshes $\mathcal{T}_h$ as $h \to 0$. Since the focus of this thesis is not on the convergence of the finite element method itself, which is readily established, but on the size of approximations the notation $u_N$ is adopted here, where $N$ is usually in a reciprocal relationship with $h$.

To be able to compute approximate solutions, a basis of $\mathcal{X}_N$ needs to be specified. Let this basis be denoted by $(\Phi_i)_{1\leq i\leq N}$, and $u_N$ in this basis given by the coefficient vector $\boldsymbol{u}$, i.e. $u_N = \sum_{i=1}^{N}[\boldsymbol{u}]_i\,\Phi_i = \boldsymbol{\Phi}^T\boldsymbol{u}$ with $\boldsymbol{\Phi}(x) = \Big[\Phi_1(x),\ldots,\Phi_N(x)\Big]^T$. Since it is sufficient that Eq. (3.2) is fulfilled on all basis vectors $\Phi_i$ of $\mathcal{X}_N$, one requires

$$a\left(\sum_{i=1}^{N}[\boldsymbol{u}]_i\,\Phi_i, \Phi_{i'}\right) = b(\Phi_{i'}), \qquad 1 \leq i' \leq N. \tag{3.5}$$

Setting $\tilde{a}(\boldsymbol{u},\boldsymbol{v}) = a(\boldsymbol{\Phi}^T\boldsymbol{u}, \boldsymbol{\Phi}^T\boldsymbol{v})$ a bilinear form on $\mathbb{R}^N \times \mathbb{R}^N$ is obtained, which can be expressed as $\tilde{a}(\boldsymbol{u},\boldsymbol{v}) = \boldsymbol{v}^T\boldsymbol{K}\boldsymbol{u}$ with $[\boldsymbol{K}]_{ii'} = a(\Phi_i, \Phi_{i'})$. Analogously, by setting $\tilde{b}(\boldsymbol{v}) = b(\boldsymbol{\Phi}^T\boldsymbol{v})$ one obtains a linear form on $\mathbb{R}^N$, which can be expressed as $\tilde{b}(\boldsymbol{v}) = \boldsymbol{v}^T\boldsymbol{f}$ with $[\boldsymbol{f}]_{i'} = b(\Phi_{i'})$. Now, as $\tilde{a}(\boldsymbol{u},\boldsymbol{v}) = \tilde{b}(\boldsymbol{v})$ for all $\boldsymbol{v} \in \mathbb{R}^N$, it follows immediately that

$$\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f}. \tag{3.6}$$

This is a sparse linear system which can be solved with standard methods of numerical linear algebra [36, 46, 89].

### 3.1.1 Stochastic variational formulation

For stochastic boundary value problems the deterministic fields $\kappa$ and $f$ turn into random fields, and thus also the solution $u$ (see e.g. [61]). Eq. (3.1) then becomes

$$-\nabla\cdot(\kappa(x,\omega)\nabla u(x,\omega)) = f(x,\omega), \qquad x \in \mathcal{D}, \tag{3.7}$$

where $\omega$ is an elementary event from a sample space $\Omega$. The sample space $\Omega$ together with the sigma algebra $\mathcal{F} \subset 2^\Omega$ and

probability measure $\mathbb{P} : \mathcal{F} \to [0, 1]$ form a complete probability space. As the sigma algebra and the probability measure are usually fixed, we will mostly omit explicit mentioning them in the following. Eq. (3.7) can now be turned into variational form namely

$$a(u, v) = b(v), \qquad \forall v \in \mathcal{V} \tag{3.8}$$

by setting $\mathcal{S} = L_2(\Omega, \mathcal{F}, \mathbb{P})$ and choosing as ansatz space $\mathcal{V} = \mathcal{X} \otimes \mathcal{S} = H^1_E \otimes L_2(\Omega, \mathcal{F}, \mathbb{P})$ with the standard inner product on tensor product Hilbert spaces

$$\langle u|v\rangle_{H^1_0 \otimes L_2(\Omega,\mathcal{F},\mathbb{P})} = \mathbb{E}\left[\int_{\mathcal{D}} u(x,\cdot)\, v(x,\cdot)\mathrm{d}x\right] \tag{3.9}$$

$$= \int_\Omega \int_{\mathcal{D}} u(x,\omega)\, v(x,\omega)\, \mathrm{d}x\, \mathrm{d}\mathbb{P}(\omega). \tag{3.10}$$

The bilinear form then becomes

$$a(u, v) = \int_\Omega \int_{\mathcal{D}} \nabla u(x,\omega)\ \cdot \kappa(x,\omega) \nabla v(x,\omega)\, \mathrm{d}x\, \mathrm{d}\mathbb{P}(\omega) \tag{3.11}$$

and the right hand side

$$b(v) = \int_\Omega \int_{\mathcal{D}} f(x,\omega) v(x,\omega)\, \mathrm{d}x\, \mathrm{d}\mathbb{P}(\omega). \tag{3.12}$$

The random field $f$ is usually required to be from $H^{-1}(\mathcal{D}) \otimes L_2(\Omega)$ and $\kappa$ from $L_\infty(\mathcal{D}) \otimes L_\infty(\Omega)$. If additionally

$$0 < \kappa_{\min} \leq \kappa(x,\omega) \tag{3.13}$$

almost everywhere in $\mathcal{D} \times \Omega$, it can be shown that the variational equation has a unique solution in $H^1_E(\mathcal{D}) \otimes L_2(\Omega)$ (see e.g. [45, 62]).

### 3.1.2 The Stochastic Galerkin method

Application of Galerkin's principle to Eq. (3.8) is in principle the same as in the deterministic case [61]. A subspace of $\mathcal{V}$ is chosen and Eq. (3.8) is required to hold only in that subspace. As

$\mathcal{V} = \mathcal{X} \otimes \mathcal{S}$ has tensor product structure, generally, though not necessarily, also a tensor product space $\mathcal{X}_N \otimes \mathcal{S}_M$ is chosen for the finite dimensional approximation. Let $(\Psi_j)_{1 \le j \le M}$ denote a basis of $\mathcal{S}_M$, then $\Phi_i \otimes \Psi_j$ is a basis of $\mathcal{X}_N \otimes \mathcal{S}_M$. Further, if the bases of $\mathcal{X}_N$ and $\mathcal{S}_M$ can be extended to countable Hilbert bases of $\mathcal{X}$ and $\mathcal{S}$, respectively, then the $\Phi_i \otimes \Psi_j$ also form a countable Hilbert basis of $\mathcal{X} \otimes \mathcal{S}$. Let $\chi_k$ denote the same tensor product basis, i.e. $\chi_k = \Phi_i \otimes \Psi_j$ for some $i$ and $j$. This could be related by some linear indexing scheme e.g. $k = \tau(i,j)$ with $\tau = N(j-1)+i$. The approximate solution $u_{NM}$ in this basis is given by the coefficient vector $\boldsymbol{u}$, i.e. $u_{NM} = \sum_{i,j=1}^{N,M} [\boldsymbol{u}]_{\tau(i,j)} \, \Phi_i \otimes \Psi_j$. Now Eq. (3.8) must be fulfilled on all basis vectors $\Phi_i \otimes \Psi_j$ of $\mathcal{X}_N \otimes \mathcal{S}_M$, requiring

$$\begin{aligned} a\left(u_{NM}, \Phi_{i'} \otimes \Psi_{j'}\right) &= a\left(\sum_{i=1}^{N}\sum_{j=1}^{M} [\boldsymbol{u}]_{\tau(i,j)} \, \Phi_i \otimes \Psi_j, \Phi_{i'} \otimes \Psi_{j'}\right) \\ &= b(\Phi_{i'} \otimes \Psi_{j'}), \end{aligned} \tag{3.14}$$

for all $i'$ and $j'$ with $1 \le i' \le N$ and $1 \le j' \le M$. Setting $[\boldsymbol{K}]_{\tau(i,j)\tau(i',j')} = a(\Phi_i \otimes \Psi_j, \Phi_{i'}\Psi_{j'})$ and $[\boldsymbol{f}]_{\tau(i',j')} = b(\Phi_{i'} \otimes \Psi_{j'})$ and collecting those $NM$ equations leads to the linear system

$$\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f}, \tag{3.15}$$

where $\boldsymbol{K}$ is now a huge $NM \times NM$ matrix.

### Variational formulation in tensor product form

The notion of a matrix in Eq. (3.15) is not optimal, even if considered as a block matrix (see [74]), since the operator has tensor product structure, which can be exploited numerically. A formulation that takes this structure into account shall be developed in the following.

Let $\kappa$ have a separable expansion of the form

$$\kappa(x,\omega) = \sum_{k=1}^{\infty} \kappa_k(x)\xi_k(\omega)$$

$$= \sum_{k=1}^{\infty} (\kappa_k \otimes \xi_k)(x, \omega) \tag{3.16}$$

where the $\kappa_k$ are pure spatial functions and the $\xi_k$ are random variables. In practice, expansions of this kind most often come from a Karhunen-Loève expansion (see Section 3.3) or from a polynomial chaos expansion (see Section 3.2).

Since $u$ is an element of a tensor product of Hilbert spaces it also has a convergent expansion

$$u = \sum_{l=1}^{\infty} u_l \otimes \eta_l. \tag{3.17}$$

Then the bilinear form in Eq. (3.11) can be formally expressed as

$$\begin{aligned} a\left(u, v \otimes \zeta\right) &= a\left(\sum_{l=1}^{\infty} u_l \otimes \eta_l, v \otimes \zeta\right) \\ &= \sum_{k=1}^{\infty} a_k^{\mathcal{X}}\left(\sum_{l=1}^{\infty} u_l, v\right) a_k^{\mathcal{S}}\left(\sum_{l=1}^{\infty} \eta_l, \zeta\right), \end{aligned} \tag{3.18}$$

where

$$a_k^{\mathcal{X}}(u, v) = \int_{\mathcal{D}} \nabla u(x) \cdot \kappa_k(x) \nabla v(x) \, \mathrm{d}x \tag{3.19}$$

and

$$a_k^{\mathcal{S}}(\eta, \zeta) = \int_{\Omega} \eta(\omega) \, \zeta(\omega) \, \xi_k(\omega) \, \mathrm{d}\mathbb{P}(\omega). \tag{3.20}$$

In general, the order of integration and summation in Eq. (3.18) cannot be exchanged, as the series expansion of $\kappa$ usually only converges in $L_2(\mathcal{D} \times \Omega)$ and the factors $\nabla u \cdot \nabla v$ and $\eta \, \zeta$ in the integrals in Eq. (3.19) and Eq. (3.20) are only guaranteed to be in $L_1(\mathcal{D})$ and $L_1(\Omega)$, respectively. However, the function spaces $\mathcal{X}_N$ and $\mathcal{S}_M$ typically have higher regularity than $\mathcal{X}$ and $\mathcal{S}$, so that when restricted to $\mathcal{X}_N \otimes \mathcal{S}_M$ the argument can be made rigorous there. For example, it is sufficient that $\nabla u \in L_4(\mathcal{D})$ for all $u \in \mathcal{X}_N$, which is the case for the usual finite element spaces,

and $\eta \in L_4(\Omega)$ for all $\eta \in \mathcal{S}_M$, which is also fulfilled for arbitrary polynomials in Gaussian random variables.

Setting $\tilde{a}(\boldsymbol{u} \otimes \boldsymbol{v}, \boldsymbol{u}' \otimes \boldsymbol{v}') = a((\boldsymbol{\Phi}^T\boldsymbol{u}) \otimes (\boldsymbol{\Psi}^T\boldsymbol{v}), (\boldsymbol{\Phi}^T\boldsymbol{u}') \otimes (\boldsymbol{\Psi}^T\boldsymbol{v})')$ defines a bilinear form on $\mathbb{R}^N \otimes \mathbb{R}^M$, which can be expressed by

$$\tilde{a}(\boldsymbol{u} \otimes \boldsymbol{v}, \boldsymbol{u}' \otimes \boldsymbol{v}') = (\boldsymbol{u}' \otimes \boldsymbol{v}')^T \, \mathbf{K} \, (\boldsymbol{u} \otimes \boldsymbol{v}) \tag{3.21}$$

where $\mathbf{K}$ is a linear map from $\mathbb{R}^N \otimes \mathbb{R}^M$ to $\mathbb{R}^N \otimes \mathbb{R}^M$. As the space of linear maps between tensor product spaces is isomorphic to a tensor product of spaces of linear maps (see Section 2.3 on page 16), there must be a representation of $\mathbf{K}$ in tensor product form.

To this end, define bilinear forms on $\mathbb{R}^N$ and on $\mathbb{R}^M$ by $\tilde{a}_k^{\mathcal{X}}(\boldsymbol{u}, \boldsymbol{u}') = \boldsymbol{u}'^T \boldsymbol{K}_k \boldsymbol{u}$ with $[\boldsymbol{K}_k]_{ii'} = a_k^{\mathcal{X}}(\Phi_i, \Phi_{i'})$ and $\tilde{a}_k^{\mathcal{S}}(\boldsymbol{v}, \boldsymbol{v}') = \boldsymbol{v}'^T \boldsymbol{\Delta}_k \boldsymbol{v}$ with $[\boldsymbol{\Delta}_k]_{jj'} = a_k^{\mathcal{S}}(\Psi_j, \Psi_{j'})$, respectively. Then the bilinear form $\tilde{a}$ can also be written as

$$\begin{aligned}
\tilde{a}(\boldsymbol{u} \otimes \boldsymbol{v}, \boldsymbol{u}' \otimes \boldsymbol{v}') &= \sum_{k=1}^{\infty} \tilde{a}_k^{\mathcal{X}}(\boldsymbol{u}, \boldsymbol{u}') \tilde{a}_k^{\mathcal{S}}(\boldsymbol{v}, \boldsymbol{v}') \\
&= \sum_{k=1}^{\infty} (\boldsymbol{u}'^T \boldsymbol{K}_k \boldsymbol{u})(\boldsymbol{v}'^T \boldsymbol{\Delta}_k \boldsymbol{v}) \\
&= \sum_{k=1}^{\infty} (\boldsymbol{u}'^T \otimes \boldsymbol{v}'^T)(\boldsymbol{K}_k \otimes \boldsymbol{\Delta}_k)(\boldsymbol{u} \otimes \boldsymbol{v}).
\end{aligned} \tag{3.22}$$

As the linear operators in Eq. (3.21) and Eq. (3.22) define the same bilinear form they can be identified, i.e.

$$\mathbf{K} = \sum_{k=1}^{\infty} (\boldsymbol{K}_k \otimes \boldsymbol{\Delta}_k) \, . \tag{3.23}$$

If the discretisation $\mathbf{u}$ of $u$ is now not regarded as a thin vector, but as an element of $\mathbb{R}^N \otimes \mathbb{R}^M$, and the same for the right hand side $\mathbf{f}$ then

$$\mathbf{K}\mathbf{u} = \sum_{k=1}^{\infty} (\boldsymbol{K}_k \otimes \boldsymbol{\Delta}_k) \, \mathbf{u} = \mathbf{f}. \tag{3.24}$$

Interpreting the tensor product in Eq. (3.24) as the Kronecker

product (see Section 4.1.3) and $\mathbf{u}$ would lead exactly back to Eq. (3.15). Leaving the equation in abstract tensor notation (the upright font here indicates this; see also Section A.2), however, allows for more choice in representation, as will be shown in Section 4.

To be able to use Eq. (3.24) in numerical computations, it is necessary to truncate the infinite series. If the expansion in Eq. (3.16) is the polynomial chaos expansion, it can be shown that only finitely many terms in the series Eq. (3.23) are non-zero anyway [62]. Otherwise, as the discrete operator $\mathbf{K}$ maps between finite dimensional spaces, it is only finite dimensional itself and the series Eq. (3.23) converges uniformly. It can thus be truncated such that the finite sums are still positive definite with uniformly bounded inverse [62].

### Higher order tensor representation

If the random field of the diffusion coefficient $\kappa$ is stochastically independent of the loading $f$, it may be worthwhile to consider a separation into third order tensor products. In this case the stochastic variational space for $u$ can be written as a tensor product $\mathcal{S} = \mathcal{S}^{(1)} \otimes \mathcal{S}^{(2)}$ where $\kappa \in \mathcal{S}^{(1)} = L_2(\Omega^{(1)}, \mathcal{F}^{(1)}, \mathbb{P}^{(1)})$ and $f, g \in \mathcal{S}^{(2)} = L_2(\Omega^{(2)}, \mathcal{F}^{(2)}, \mathbb{P}^{(2)})$. The full stochastic variational space is then $\mathcal{S} = L_2(\Omega, \mathcal{F}, \mathbb{P}) \simeq L_2(\Omega^{(1)}, \mathcal{F}^{(1)}, \mathbb{P}^{(1)}) \otimes L_2(\Omega^{(2)}, \mathcal{F}^{(2)}, \mathbb{P}^{(2)})$, where the sample space $\Omega = \Omega^{(1)} \times \Omega^{(2)}$ is now the Cartesian product of the two sample spaces $\Omega^{(1)}$ and $\Omega^{(2)}$, $\mathcal{F} = \mathcal{F}^{(1)} \times \mathcal{F}^{(2)}$ is the product sigma algebra and $\mathbb{P} = \mathbb{P}^{(1)} \otimes \mathbb{P}^{(2)}$ is the (completed) product measure. This is due to the isometry of tensor products of $L_2$ spaces as seen in Section 2.2.

In order to apply Galerkin's principle, subspaces $\mathcal{S}^{(1)}_{M^{(1)}} \subset \mathcal{S}^{(1)}$ and $\mathcal{S}^{(2)}_{M^{(2)}} \subset \mathcal{S}^{(2)}$ for the discretisation can be defined independently, so that the complete stochastic ansatz space becomes $\mathcal{S}_M = \mathcal{S}^{(1)}_{M^{(1)}} \otimes \mathcal{S}^{(2)}_{M^{(2)}}$ and $\mathcal{X}_N \otimes \mathcal{S}^{(1)}_{M^{(1)}} \otimes \mathcal{S}^{(2)}_{M^{(2)}}$ the full ansatz space. Choosing bases $(\Psi^{(l)}_j)_{1 \leq j \leq M^{(l)}}$ for the spaces $\mathcal{S}^{(l)}_{M^{(l)}}$ $(l = 1, 2)$ Eq. (3.7) can be discretised in the same manner as in Section 3.1.2

Noting that the stochastic bilinear form Eq. (3.20) now splits

into a product of two integrals

$$a_k^{\mathcal{S}}(\eta^{(1)} \otimes \eta^{(2)}, \zeta^{(1)} \otimes \zeta^{(2)}) = \int_{\Omega^{(1)}} \eta^{(1)}(\omega)\, \zeta^{(1)}(\omega)\, \xi_k(\omega)\, \mathrm{d}\mathbb{P}^{(1)}(\omega) \times \int_{\Omega^{(2)}} \eta^{(2)}(\omega)\, \zeta^{(2)}(\omega)\, \mathrm{d}\mathbb{P}^{(2)}(\omega),$$

the stochastic Galerkin matrices $\boldsymbol{\Delta}_k$ can be written as

$$\boldsymbol{\Delta}_k = \boldsymbol{\Delta}_k^{(1)} \otimes \boldsymbol{G}_\Psi^{(2)}$$

where

$$[\boldsymbol{\Delta}_k^{(1)}]_{jj'} = a_k^{\mathcal{S}}(\Psi_j^{(1)}, \Psi_{j'}^{(1)})$$

and

$$[\boldsymbol{G}_\Psi^{(2)}]_{ll'} = \int_{\Omega^{(2)}} \Psi_l^{(2)}(\omega)\, \Psi_{l'}^{(2)}(\omega)\, \mathrm{d}\mathbb{P}^{(2)}(\omega)$$

is the Gramian matrix for the basis $(\Psi_j^{(2)})_{1 \le j \le M^{(2)}}$ of $\mathcal{S}_{M^{(2)}}^{(2)}$. The fully discretised equation can then be written as

$$\left( \sum_{k=0}^{L} \boldsymbol{K}_k \otimes \boldsymbol{\Delta}_k^{(1)} \otimes \boldsymbol{G}_\Psi^{(2)} \right) \mathbf{u} = \mathbf{f}. \tag{3.25}$$

Here, $\mathbf{u}$ and $\mathbf{f}$ are now third order tensors. The loading $\mathbf{f}$ can be easily constructed from the second order tensor representation of $\mathbf{f}$ by the following embedding from $\mathcal{X} \otimes \mathcal{S}^{(2)} \to \mathcal{X} \otimes \mathcal{S}^{(1)} \otimes \mathcal{S}^{(2)}$, namely $\sum_i f_i \otimes \xi_i^{(2)} \mapsto \sum_i f_i \otimes 1 \otimes \xi_i^{(2)}$.

The advantage of this form is that the representation of $\mathbf{f}$ and $\mathbf{u}$ and of the stochastic operator can be made cheaper in terms of memory usage, and further that the application of the operator becomes more efficient, as the matrix $\boldsymbol{G}_\Psi^{(2)}$ is diagonal for orthogonal bases. However, as will be discussed in Section 4.2, tensor truncations for third order tensors are generally not as efficient as for second order tensors. Hence, computations in this format are generally not as efficient in terms of runtime as for second order tensors.

## 3.2 The Polynomial Chaos Expansion

For the choice of the subspace $\mathcal{S}_M$ and basis functions $\psi_j$ different methods have been developed, some of which shall be reviewed in this section. Since the problems considered in this thesis pose no particular difficulties for the stochastic discretisation, we stick here with the classical expansion of random variables with finite variance into an $L_2$-convergent series, the polynomial chaos expansion. This expansion goes back to Wiener [90] and was first employed in numerical methods for uncertainty quantification by Ghanem and Spanos [28]. As the approach has been described numerously elsewhere (see e.g. [42, 45, 56]), only a brief sketch shall be given here.

Let $H \subset L_2(\Omega, \mathcal{F}, \mathbb{P})$ be a separable Gaussian Hilbert space and $X \in L_2(\Omega, \sigma(H), \mathbb{P})$ a random variable, that is measurable with respect to the sigma algebra $\sigma(H)$ generated by $H$. Let $\boldsymbol{\theta} = (\theta_1, \theta_2, \dots, )$ be a sequence of independent zero mean, unit variance Gaussian random variables, and let $\mathcal{J} = (\mathbb{N}_0)_c^{\mathbb{N}}$ be the set of sequences with values and in $\mathbb{N}_0$ and finite support. Elements $\alpha \in \mathcal{J}$ of this set are called multiindices and $|\alpha| = \|\alpha\|_1 = \alpha_1 + \alpha_2 + \cdots$ is called the order of the multiindex. Then $X$ can be represented by the $L_2$-convergent series

$$X = \sum_{\alpha \in \mathcal{J}} X_\alpha H_\alpha(\boldsymbol{\theta}), \tag{3.26}$$

where $H_\alpha$ denotes the multivariate Hermite polynomials, i.e. $H_\alpha(\boldsymbol{\theta}) = H_{\alpha_1}(\theta_1) H_{\alpha_2}(\theta_2) \cdot \cdots$ is the product of the univariate probabilist Hermite polynomials. Note, that in the product only a finite number of terms is not equal to one, so that the product depends only on a finite number of the random variables $\theta_i$. Due to the orthogonality of the multivariate Hermite polynomials the coefficients $X_\alpha$ in Eq. (3.26) can be computed by

$$X_\alpha = \mathbb{E}\left[X(\cdot) H_\alpha(\cdot)\right] / \mathbb{E}\left[H_\alpha^2\right]. \tag{3.27}$$

**Remark 3.1.** *One possible difficulty for the use of the polynomial*

*chaos expansion is that the polynomial chaos over some Gaussian Hilbert space $H$ may* not *be dense in $L_2(\Omega, \mathcal{F}, \mathbb{P})$, but rather in a subspace $L_2(\Omega, \sigma(H), \mathbb{P})$, where $\sigma(H) \subset \mathcal{F}$ denotes the sigma algebra generated by all random variables in $H$. Since this sigma algebra is in general coarser than $\mathcal{F}$, there may be random variables $X$ that are measurable with respect to $\mathcal{F}$, but not with respect to $\sigma(H)$. A question, that comes up naturally then, is, whether this affects computability or approximability of random quantities of interest in the intended applications, i.e. the solution of an SPDE or functionals thereof. Luckily, the answer is no; the reason being that all those quantities are Borel measurable functions of the input random variables or fields, and thus measurable with respect to the sigma algebra generated by them. As the space $H$ is chosen (or rather constructed) in such a way that the input random quantities become measurable in $\sigma(H)$, this carries over to the quantities of interest as well.*

### 3.2.1 Choice of a finite-dimensional subspace

For numerical computations a finite subset of the polynomial chaos basis—or in other words a finite dimensional subspace of $L_2(\Omega)$—has to be chosen. Since the PC basis is indexed by the multiindex set $\mathcal{J}$, choosing a subspace is equivalent to choosing a subset $\mathcal{I} \subset \mathcal{J}$ of multiindices, and this terminology will be used interchangeably. In the literature a variety of subsets has been used, from which the most common or interesting will be briefly reviewed.

**Complete polynomials:** The polynomial chaos space is restricted to complete a polynomials in $m$ variables up to total order $p$, i.e. the multiindex order $|\alpha| = \alpha_1 + \alpha_2 + \cdots$ does not exceed $p$ (see e.g. [27, 28, 62, 86]). The size of this multiindex set $\mathcal{I}_{m,p}$ grows as

$$|\mathcal{I}_{m,p}| = \frac{(m+p)!}{m!\,p!}, \tag{3.28}$$

where the highest order terms are $m^p/p!$ for fixed $m$ and $p^m/m!$ for fixed $p$. This approach is predominant in the

literature and will also be employed in this thesis.

**Complete polynomials with a limited number of terms:** This set, denoted by $\mathcal{I}_{m,p,l}$, is a subset of $\mathcal{I}_{m,p}$, in which no more than $l$ of the $\alpha_i$ are allowed to be non-zero. However, for small $p$ the effect on size reduction is not too pronounced.

**Tensor product polynomials:** $\mathcal{I}$ is restricted to a subset of $m$ Gaussians up to polynomial order $p$ in *each* variable, giving rise to a set $\mathcal{I}^T_{m,p}$ of multiindices of size $(p+1)^m$ (see e.g. [86]). This is sometimes computationally advantageous as the tensor product structure permits easy computation of linear indices out of the multiindices, making computations e.g. of the multiplication tensor particularly easy. However, the size of the multiindex grows very fast with $m$ and $p$, so that this set is seldom employed.

**Tensor product polynomials with variable degree:** In this approach, which has been used e.g. by Ullmann [86] and by Frauenfelder et al. [25], the polynomial degree is allowed to vary and thus described by a tuple of degrees $\boldsymbol{p} = (p_1, \ldots, p_m)$. In the latter a stochastic boundary value problem was solved with $m = 30$ and with $\max_i(p_i) = 10$, where the polynomial degrees were chosen adaptively based on the decay of the KL eigenvalues of the diffusion coefficient.

**A priori adaptive reduction:** For translation random fields the Gaussian random variables are usually ordered by decreasing mode strength in the underlying Gaussian random field. In the approaches described above all Gaussians were expanded up to the same polynomial order without taking the relative mode strengths into consideration. In the framework of stochastic collocation methods Nobile et al. (see [65]) have used this to construct anisotropic sparse grids and showed some strict error bounds. This approach can also be translated to stochastic Galerkin methods, as the sparse grids can be seen to directly correspond to some polynomial

interpolation. However, no strict results have been derived in this case as yet. Furthermore, while this probably allows more efficient approximation of the input fields, it is not clear how good this is for the approximation of the solution.

**Adaptive reduction:** El Moselhy and Marzouk [21] have developed an adaptive strategy, which selects the best subspace out of a huge initial space given by complete polynomials in a large number of random variables ($m \approx 500$) and up to high order ($p \approx 14$). They could show that usually a very small subspace of dimension $\approx 2000$ is sufficient to achieve the required accuracy. An interesting aspect of their results is that, while higher modes usually appear only in relatively low order, they often appear not on their own, but as factors of polynomials of higher order. The method needs certain integrals to be known analytically and is thus tailored to some particular input fields (log-normal) for which those integrals are known. Whether extensions to arbitrary input fields are possible is not yet resolved.

## 3.3 The Karhunen-Loève Expansion

One of the most important tools in the numerical treatment of stochastic PDEs is the Karhunen-Loève expansion [44, 58]. A brief review of the properties of the KLE for sufficiently well behaved random fields shall be given here; more details can be found e.g. in [28, 61].

Given a random field $r(x, \omega) \in L_2(\mathcal{D}) \otimes L_2(\Omega)$ the KLE is the representation of $r$ as the infinite sum

$$r(x, \omega) = \bar{r}(x) + \sum_{i=1}^{\infty} \sigma_i \, r_i(x) \, \xi_i(\omega) \tag{3.29}$$

where $\bar{r}(x) = \mathbb{E}\left[r(x, \cdot)\right]$ is the mean value of the random field and $\sigma_0 \geq \sigma_1 \geq \sigma_2 \geq \cdots$ is a decreasing, non-negative sequence of real numbers with limit point 0. The $r_i$ form an orthonormal system

in $L_2(\mathcal{D})$, and the $\xi_i$ are uncorrelated random variables in $L_2(\Omega)$ of unit variance.

**Remark 3.2.** *Often the KLE is written in the more compact form*

$$r(x,\omega) = \sum_{i=0}^{\infty} \sigma_i \, r_i(x) \, \xi_i(\omega), \tag{3.30}$$

*where* $\sigma_0$, $r_0(x) = \bar{r}(x)$ *and* $\xi_0(\omega) = 1$, *which will be done frequently in this work.*

The importance of the KLE stems from the following facts:

- The KLE gives a representation of the random field in a *countable* number of random variables.
- The random variables are uncorrelated, which in the case of Gaussian random fields, implies independence.
- The truncation of the KLE after a finite number of terms, say $L$, gives the best approximation in variance of the random field $r$ in $L$ terms.
- The spatial part of the expansion can be computed from second order statistical information on the random field alone.

The computation of the KLE, which also proves its existence, can be done in the following way. Define the operator

$$C : L_2(\mathcal{D}) \to L_2(\mathcal{D}) \tag{3.31}$$

$$v \mapsto \int_{\mathcal{D}} \operatorname{cov}_r(\cdot, y) v(y) \, \mathrm{d}y \tag{3.32}$$

where $\operatorname{cov}_r(x,y) = \mathbb{E}\left[(r(x,\cdot) - \bar{r}(x))(r(y,\cdot) - \bar{r}(y))\right]$ is the covariance function of $r$. The $\sigma_i$ and the $r_i$ are solutions to the eigenvalue problem

$$C r_i = \sigma_i^2 r_i. \tag{3.33}$$

If $\operatorname{cov}_r \in L_2(\mathcal{D} \times \mathcal{D})$, then $C$ is a symmetric, positive-definite Hilbert-Schmidt operator and has a countable number of eigenvalues. The eigenfunctions of $C$ are mutually orthogonal and form

a basis of $L_2(\mathcal{D})$ (for details see [45] and the references therein). Using the orthogonality of the KL eigenfunctions the random variables $\xi_i$ in Eq. (3.29) can be computed by projection

$$\xi_i(\omega) = \int_{\mathcal{D}} (r(x,\omega) - \bar{r}(x))\, r_i(x)\, \mathrm{d}x. \tag{3.34}$$

Further for the KL eigenvalues it holds that

$$\sum_{i=1}^{\infty} \sigma_i^2 = \|r - \bar{r}\|^2_{L_2(\mathcal{D}) \otimes L_2(\Omega)}, \tag{3.35}$$

and for the truncated KLE with $L$ terms

$$\sum_{i=L+1}^{\infty} \sigma_i^2 = \|r_L - r\|^2_{L_2(\mathcal{D}) \otimes L_2(\Omega)}. \tag{3.36}$$

which can be employed for estimating the error in truncating the KLE in practice. If the random field $r$ has constant variance $\sigma^2$, it follows immediately that

$$\|r_L - r\|^2_{L_2(\mathcal{D}) \otimes L_2(\Omega)} = |\mathcal{D}|\sigma^2 - \sum_{i=1}^{L} \sigma_i^2, \tag{3.37}$$

where $|\mathcal{D}|$ is the area of the domain.

**Remark 3.3.** *Note that while the KL eigenvalues and eigenfunctions can be computed from second order statistics alone, for the computation of the KL random variables the field must be completely specified. Typically, this is only the case, if the field is Gaussian or a function of a Gaussian field, or some special type of field that does not need the KLE for approximation anyway (like e.g. zonal approximations in geostatistical modelling [70]). Notwithstanding, the term KLE is often used in the literature misleadingly for the solution of the eigenvalue problem alone, and the random variables are not computed by (3.34), but specified in some rather ad-hoc fashion.*

Analytical solutions to Eq. (3.33) are known in only a few cases

[28]. For most problems in higher dimensions numerical solutions have to be sought [4, 45].

There is a natural connection between the KLE and the singular value decomposition as will be shown in the case of discretised random fields in Section 3.3.2.

### 3.3.1 Numerical computation of the KLE

Methods for the numerical computation of the KLE depend on whether the expansion needs to be computed from the covariance for the input random fields or whether it has to be computed for a random field that is already discretised. For the former a variety of different methods exist. Standard Galerkin methods are discussed in [45], $\mathcal{H}$-matrix based methods in [47], fast multipole methods in [25]—just to name a few. As this part is insignificant for this thesis, standard Galerkin type approximations as discussed in [45] have been used. These may not be as efficient as some of the advanced methods, but are stable and straightforward to implement.

For the latter, SVD based methods can be used, which will be shown in the next section (see also tensor truncation in Section 4.1.5). Methods based on the covariance function of the discretised field have also been employed in some publications (see e.g. [20]). In the author's view, this is not optimal since it negatively affects the condition of the problem, and is not as efficient as SVD based methods.

### 3.3.2 The KLE and the SVD

The singular value decomposition [3, 32, 84] for a given real matrix $\boldsymbol{A} \in \mathbb{R}^{N \times M}$ is the product

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T, \tag{3.38}$$

where $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthogonal and $\boldsymbol{\Sigma}$ is a diagonal matrix with monotone decreasing positive entries. If the columns of $\boldsymbol{U}$ and $\boldsymbol{V}$ are denoted by $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$, Eq. (3.38) can also be written in the

form

$$\boldsymbol{A} = \sum_{i=1}^{n} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T, \tag{3.39}$$

where $n = \min(N, M)$. From the Eckart-Young theorem it is well known that truncating the sum in Eq. (3.39) after $L$ terms gives the best rank-$L$ approximation to $\boldsymbol{A}$. In other words,

$$\boldsymbol{A}_L = \sum_{i=1}^{L} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T, \tag{3.40}$$

minimises $|||\boldsymbol{A} - \boldsymbol{A}_L|||$ over all matrices of rank $L$, where $|||\cdot|||$ is any unitarily invariant matrix norm.

It is apparent from the properties of the SVD and the form of Eq. (3.39) on the one hand, and from the properties of the KLE and Eq. (3.29) on the other, that the SVD is the discrete form of the KLE. While it is a classical result that the KLE of a random field $r$ is equivalent to the SVD of the integral operator $R : \varphi \mapsto \int_{\mathcal{D}} r(x, \cdot)\varphi(x)dx$ (see e.g. [53]), the exact connection between discretised random fields and the matrix SVD of its coefficient matrix and its implications for practical computations have not been studied so far. In the following this connection shall thus be elucidated in more detail.

A discretised random field $r$ in $L_2(\mathcal{D}) \otimes L_2(\Omega)$ can be written as

$$r(x, \omega) = \boldsymbol{\Phi}(x)^T \boldsymbol{R} \boldsymbol{\Psi}(\omega) \tag{3.41}$$

where $\boldsymbol{\Phi}(x)$ and $\boldsymbol{\Psi}(\omega)$ are the vectors of spatial and stochastic basis functions and $\boldsymbol{R}$ is the matrix of coefficients. For ease of presentation $r$ is assumed to be zero-mean.

Now, the spatial KL eigenfunctions $r_k$ of r can be written as

$$r_k(x) = \sum_{i=1}^{N} s_{ik} \Phi_i(x) = \left[\boldsymbol{\Phi}(x)^T \boldsymbol{S}\right]_k, \tag{3.42}$$

where $[\boldsymbol{S}]_{ik} = s_{ik}$ and the KL random variables $\xi_k$ can be written

as

$$\xi_k(\omega) = \sum_{j=1}^{M} \xi_{kj} \Psi_j(\omega) = \left[ \boldsymbol{\Psi}(\omega)^T \boldsymbol{\Xi} \right]_k, \tag{3.43}$$

where $[\boldsymbol{\Xi}]_{jk} = \xi_{jk}$. Inserting this into Eq. (3.29) gives

$$\begin{aligned} r(w,\omega) &= \sum_{k=1}^{n} \left[ \boldsymbol{\Phi}(x)^T \boldsymbol{S} \right]_k \sigma_k \left[ \boldsymbol{\Psi}(\omega)^T \boldsymbol{\Xi} \right]_k \\ &= \boldsymbol{\Phi}(x)^T \boldsymbol{S} \boldsymbol{\Sigma} \boldsymbol{\Xi}^T \boldsymbol{\Psi}(\omega) \end{aligned} \tag{3.44}$$

Since the basis functions are linearly independent, it follows immediately that $\boldsymbol{S}\boldsymbol{\Sigma}\boldsymbol{\Xi}^T$ must be a decomposition of $\boldsymbol{R}$. Further, from the orthogonality of the KL eigenfunctions and random variables follows

$$\begin{aligned} I &= \langle \boldsymbol{\Phi}(\cdot)\boldsymbol{S} | \boldsymbol{\Phi}(\cdot)\boldsymbol{S} \rangle_{L_2(\mathcal{D})} \\ &= \boldsymbol{S}^T \left( \int_{\mathcal{D}} \boldsymbol{\Phi}(x) \boldsymbol{\Phi}(x)^T \, \mathrm{d}x \right) \boldsymbol{S} \\ &= \boldsymbol{S}^T \boldsymbol{G}_{\boldsymbol{\Phi}} \boldsymbol{S}, \end{aligned} \tag{3.45}$$

and

$$\begin{aligned} I &= \langle \boldsymbol{\Psi}(\cdot)\boldsymbol{\Xi} | \boldsymbol{\Psi}(\cdot)\boldsymbol{\Xi} \rangle_{L_2(\omega)} \\ &= \boldsymbol{\Xi}^T \left( \int_{\Omega} \boldsymbol{\Psi}(\omega) \boldsymbol{\Psi}(\omega)^T \, \mathrm{d}\mathbb{P}(\omega) \right) \boldsymbol{\Xi} \\ &= \boldsymbol{\Xi}^T \boldsymbol{G}_{\boldsymbol{\Psi}} \boldsymbol{\Xi}, \end{aligned} \tag{3.46}$$

i.e. $\boldsymbol{S}$ is orthogonal with respect to the inner product $\langle \boldsymbol{u} | \boldsymbol{v} \rangle_{\boldsymbol{G}_{\boldsymbol{\Phi}}} = \boldsymbol{u}^T \boldsymbol{G}_{\boldsymbol{\Phi}} \boldsymbol{v}$, where

$$\boldsymbol{G}_{\boldsymbol{\Phi}} = \int_{\mathcal{D}} \boldsymbol{\Phi}(x) \boldsymbol{\Phi}(x)^T \, \mathrm{d}x \tag{3.47}$$

is the spatial Gram matrix, and $\boldsymbol{\Xi}$ is orthogonal with respect to

the inner product $\langle \boldsymbol{\phi} | \boldsymbol{\psi} \rangle_{\boldsymbol{G_\Psi}} = \boldsymbol{\phi}^T \boldsymbol{G_\Psi} \boldsymbol{\phi}$, where

$$\boldsymbol{G_\Psi} = \mathbb{E}\left[ \boldsymbol{\Psi}(\cdot) \boldsymbol{\Psi}(\cdot)^T \right] \tag{3.48}$$

is the stochastic Gram matrix. The product $\boldsymbol{S \Sigma \Xi}^T$ can thus be regarded as a generalised singular value decomposition (GSVD)[1] of $\boldsymbol{R}$, with orthogonality defined by the scalar products $\langle \cdot | \cdot \rangle_{\boldsymbol{G_\Phi}}$ and $\langle \cdot | \cdot \rangle_{\boldsymbol{G_\Psi}}$.

**Remark 3.4.** *In some works a generalised Karhunen-Loève expansion, called the Hilbert-Karhunen-Loève expansion, is used in which the spatial inner product is adapted to the problem domain [20]. For example, for elliptic SPDEs with solutions in $H^1$, it may be preferable to use the $H^1$ inner product defined by $\langle u | v \rangle_{H^1} = \int_{\mathcal{D}} \nabla u \cdot \nabla v \mathrm{d}x$. This can be easily accomplished with the method presented here by using $[\boldsymbol{G_\Phi}]_{ij} = \int_{\mathcal{D}} \nabla \Phi(x) \cdot \nabla \Phi(x) \, \mathrm{d}x$ as the spatial Gramian instead of Eq. (3.47).*

### Computation of the KLE for discretised random fields

To the author's knowledge, the only published approach for computing the KLE of a discretised random field as given by Eq. (3.41), is based on the diagonalisation of the covariance matrix [20]. Since this approach is numerically not optimal, a new approach based on the results of the previous section has been developed in this work.

Basically, there are two methods to compute the KLE based on the SVD approach. The first one can be used if $N$ or $M$ is small, or if the full KLE is needed. In this case one computes the Cholesky decompositions $\boldsymbol{L_\Phi}$ and $\boldsymbol{L_\Psi}$ of $\boldsymbol{G_\Phi}$ and $\boldsymbol{G_\Psi}$, which does not add significantly to the effort spent in computing the SVD. Then the SVD of the product $\boldsymbol{L_\Phi R L_\Psi}^T$ is computed by

$$\boldsymbol{U \Sigma V}^T = \boldsymbol{L_\Phi R L_\Psi}^T. \tag{3.49}$$

[1] Note that there are more ways to generalise the SVD, and thus the term is used ambiguously in the literature. The meaning here is that of a weighted PCA as given e.g. in [1, 43].

Now, solving with the Cholesky factors gives $\boldsymbol{S} = \boldsymbol{L}_{\boldsymbol{\Phi}}^{-1}\boldsymbol{U}$ and $\boldsymbol{\Xi} = \boldsymbol{L}_{\boldsymbol{\Psi}}^{-1}\boldsymbol{V}$ and $\boldsymbol{S}\boldsymbol{\Sigma}\boldsymbol{\Xi}^T$ is the SVD of $\boldsymbol{R}$ with respect to the $\boldsymbol{G}_{\boldsymbol{\Phi}}$ and $\boldsymbol{G}_{\boldsymbol{\Psi}}$ inner products, or in other words, the KLE of $r$.

If $N$ or $M$ are not small, or if only a few modes of the KLE need to be computed, it is common to use iterative, matrix-free methods for computing the SVD. Computing the Cholesky factors would be infeasible then or add significantly to the cost, so that a different approach has to be taken. Many procedures (e.g. `svds` in Matlab® [60] or `las1` in SVDPACK [11]) compute the sparse SVD of a matrix $\boldsymbol{R}$ by computing the eigensystem of the matrix [10]

$$\boldsymbol{R}' = \begin{bmatrix} 0 & \boldsymbol{R} \\ \boldsymbol{R}^T & 0 \end{bmatrix} \tag{3.50}$$

using a sparse eigenvalue solver (e.g. ARPACK [57]). Defining now the matrix

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{G}_{\boldsymbol{\Phi}} & 0 \\ 0 & \boldsymbol{G}_{\boldsymbol{\Psi}} \end{bmatrix} \tag{3.51}$$

the SVD can be computed by solving the generalised eigenvalue problem

$$(\boldsymbol{G}\boldsymbol{R}'\boldsymbol{G})\boldsymbol{w} = \lambda \boldsymbol{G}\boldsymbol{w}, \tag{3.52}$$

which can be done by matrix-free methods without explicitly forming the matrix on the left hand side. The generalised singular values of $\boldsymbol{R}$ are the positive eigenvalues $\lambda$ of Eq. (3.52) and the left and right singular vectors (or KL eigenfunctions and random variables) can be extracted from the eigenvectors $\boldsymbol{w}$. The left singular vectors correspond to the upper part and the right singular vectors to the lower part of $\boldsymbol{w}$, i.e. if $\boldsymbol{w} = \begin{bmatrix} \tilde{\boldsymbol{u}}^T & \tilde{\boldsymbol{v}}^T \end{bmatrix}^T$ is an eigenvector to the positive eigenvalue $\lambda$, then $\boldsymbol{u} = \tilde{\boldsymbol{u}}/\|\tilde{\boldsymbol{u}}\|_2$ and $\boldsymbol{v} = \tilde{\boldsymbol{v}}/\|\tilde{\boldsymbol{v}}\|_2$ are the left and right singular vectors corresponding to the singular value $\sigma = \lambda$.

In numerical application the KLE is often used to produce optimal low-rank approximations of random fields by retaining only a few modes of its KLE. It is now interesting to see, whether the quality of a $k$-term approximation $r_k$ to a random field $r$

depends on whether the GSVD, corresponding to the exact KLE, or the standard matrix SVD is employed—in other words, whether the best $k$-term approximation in the $L_2$- or in the $\ell_2$-sense is computed.

In order to analyse this, the errors in the $L_2(\mathcal{D} \times \Omega)$ norm between a random field $r$ and its truncated $k$-term KLE $\|r - r_k^{\mathrm{KL}}\|_{L_2}$, computed by the GSVD described above, and its truncated SVD expansion $\|r - r_k^{\mathrm{SVD}}\|_{L_2}$ have been measured. The random field used for the numerical experiments was the solution $u$ of the stochastic PDE using the medium model described in Section 8.1 on page 117. The measured $L_2$-errors can be seen in Fig. 3.1 (left) and the difference in the errors in Fig. 3.1 (right). The difference in the truncation errors are barely visible in the left figure, and the right figure supports this. Of course, this finding is not too surprising as the stochastic Gramian is diagonal with only small entries (between $\min_\alpha(\|H_\alpha\|)^2 = 1$ and $\max_\alpha(\|H_\alpha\|)^2 = 6$) and the spatial Gramian comes from a mesh with triangles of nearly the same size, which gives the basis functions nearly equal weights. However, the result indicates that it is valid to use the standard matrix SVD for truncation, instead of using the GSVD, because this is computationally much more efficient.

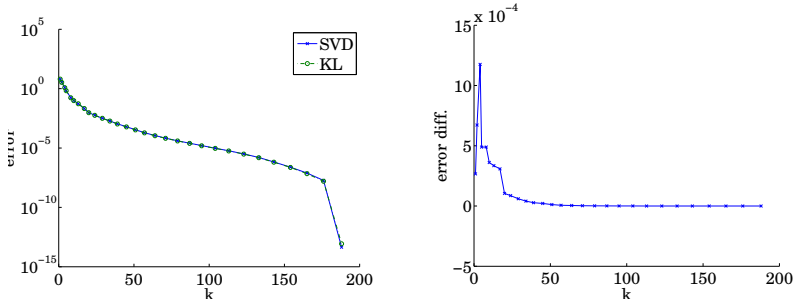



**Figure 3.1:** The left graph shows the error in the $L_2$ norm for different truncation ranks $k$ using SVD-based and KLE-/GSVD-based truncation. The right graph show the difference between the $L_2$-errors incurred. (Script: `figures/show_svd_vs_kl_error`)

**Remark 3.5.** *If the $H^1$-norm, instead of the $L_2$-norm, or adap-*

*tively refined grids are used, it has to be seen whether this observation still holds or the true KLE is better computed according to the GSVD algorithm outlined above.*

## 3.4 Discretisation of random fields

For numerical tests of the methods developed in this thesis random fields as inputs for the right hand side and for the conductivity had to be generated. In general, to fully describe a random field all finite joint distributions (fidi) must be specified (see e.g. [45]). For practical reasons, however, it was assumed that only the marginal distributions and second order statistics, i.e. the covariance, of the field are available. Since this data is not sufficient to fully specify the random field, the common assumption is made that the field is a nonlinearly transformed Gaussian random field.

**Remark 3.6.** *This method is also known as NORTA method, which is an abbreviation for "NORmal To Anything". Here, it is specifically adapted to the case of a random field given in PCE variables. The random fields constructed by this method are often called—in a slightly misleading way—translation Gaussian random fields.*

The method used here is adopted from a paper by Sakamoto and Ghanem [81]. However, we restrict ourselves here to stationary random fields, making notation a bit simpler and allowing for some optimisations, and further clarify some points left unclear in [81]. The general outline of the method is the following:

- Infer from the covariance of the target random field the covariance of the underlying Gaussian random field. Note that in some works, the covariance of the Gaussian random field is assumed to be given. This assumption is, however, quite artificial, given that generally only measurements of the target random field are available.

- Using the covariance of the Gaussian random field, compute its KLE. The random variables of the KLE are uncorrelated and jointly Gaussian, thus independent. This is all

information that needs to be known about a set of Gaussian random variables to infer every possible statistics of them. In other words, although the Gaussian random variables $\theta_i$ are in $L_2(\Omega, \mathcal{F}, \mathbb{P})$, the functional dependence on $\Omega$ can be ignored from now on, since any statistical inference, which amounts to some kind of integration over the $\theta_i$, can be directly obtained. This can also be seen as replacing the probability space $(\Omega, \mathcal{F}, \mathbb{P})$ by $(\mathbb{R}^{\mathbb{N}}, \mathcal{B}(\mathbb{R}^{\mathbb{N}}), \Gamma^{\mathbb{N}})$, where $(\theta_1, \ldots) \in \mathbb{R}^{\mathbb{N}}$, $\mathcal{B}(\mathbb{R}^{\mathbb{N}})$ is the Borel sigma algebra on $\mathbb{R}^{\mathbb{N}}$ and $\Gamma$ is the Gaussian measure.

- Via an expansion of the transform into Hermite polynomials the PCE of the target random field can be directly computed from the KLE of the Gaussian random field.

### 3.4.1 Transformation of the covariance

The following is adopted from [81] to the notation used within this thesis, and some details of the derivation have been added. Let the target random field, which is assumed to be a transformed Gaussian random field, be denoted by $r(x, \omega)$. For simplicity it is further assumed that the field is stationary, because then no explicit location dependence is needed in the transform function (and in this thesis only stationary fields are used as inputs). Let the marginal cumulative probability density function (CDF) of $r$ be given by $F_r(r') = \mathbb{P}(r(x, \omega) < r')$ and its covariance function by $\mathrm{cov}_r(x, y)$. Then the random field $r$ can be written as

$$\begin{aligned} r(x, \omega) &= F_\rho^{-1}(\Phi(\gamma(x, \omega))) \\ &= \varphi(\gamma(x, \omega)), \end{aligned} \tag{3.53}$$

where $\Phi$ is the cumulative distribution function of the standard normal distribution $\mathcal{N}(0, 1)$. For example, for a stationary log-normal random field $r \sim \ln \mathcal{N}(\mu, \sigma^2)$ the transform function is $\varphi(\gamma) = \exp(\sigma\gamma + \mu)$, for a uniformly distributed field with distribution $\mathcal{U}(a, b)$ the transform is $\varphi(\gamma) = a + (b - a)\Phi(\gamma)$. The Gaussian random field $\gamma$ is assumed to have a standard normal

distribution, since otherwise this can always be achieved by a change of the transform function.

Expanding $\varphi$ in a series in the Hermite polynomials gives

$$\varphi(z) = \sum_{i=0}^{\infty} \varphi_i H_i(z), \tag{3.54}$$

where

$$\varphi_i = \int_{\mathbb{R}} \varphi(z) H_i(z) \frac{\exp(-z^2/2)}{\sqrt{2\pi}\, i!} \mathrm{d}z. \tag{3.55}$$

Now, it follows for the covariance of the Gaussian field $\mathrm{cov}_\gamma$

$$\begin{aligned} \mathrm{cov}_r(x,y) &= \langle r(x,\cdot) | r(y,\cdot) \rangle \\ &= \sum_{i,j=0}^{\infty} \varphi_i \varphi_j \langle H_i(\gamma(x,\cdot)) | H_j(\gamma(y,\cdot)) \rangle \\ &= \sum_{i,j=0}^{\infty} \varphi_i\, \varphi_j\, i!\, \delta_{ij}\, \langle \gamma(x,\cdot) | \gamma(y,\cdot) \rangle^i \\ &= \sum_{i=0}^{\infty} i!\, \varphi_i^2\, \mathrm{cov}_\gamma(x,y)^i, \end{aligned} \tag{3.56}$$

where in the last equality the relation

$$\langle H_i(\gamma_1) | H_j(\gamma_2) \rangle = \delta_{ij}\, i!\, \langle \gamma_1 | \gamma_2 \rangle^i \tag{3.57}$$

has been used. To see this, assume without loss of generality that $i \geq j$. Further, set $\gamma_2 = \alpha\gamma_1 + \hat{\gamma}_2$ with $\alpha = \langle \gamma_1 | \gamma_2 \rangle$ and $\langle \gamma_1 | \hat{\gamma}_2 \rangle = 0$. Then

$$\begin{aligned} \langle H_i(\gamma_1) | H_j(\gamma_2) \rangle &= \langle H_i(\gamma_1) | H_j(\alpha\gamma_1 + \hat{\gamma}_2) \rangle \\ &= \alpha^j \langle H_i(\gamma_1) | H_j(\gamma_1) + p_{j-1;\hat{\gamma}_2}(\gamma_1) \rangle \end{aligned} \tag{3.58}$$

where $p_{j-1;\hat{\gamma}_2}(\gamma_1)$ is a polynomial of degree at most $j-1$ in $\gamma_1$ with coefficients depending on $\gamma_2$. The relation Eq. (3.57) then follows immediately from the orthogonality relation of the Hermite polynomials.

When the series in Eq. (3.56), is truncated after $n$ terms, solving

the equation reduces to finding the roots of a polynomial in $\mathrm{cov}_\gamma(x,y)$ with coefficients $i!\,\varphi_i^2$ for $i>0$ and $\varphi_0^2-\mathrm{cov}_r(x,y)$ for $i=0$, i.e.

$$\varphi_0^2-\mathrm{cov}_r(x,y)+\sum_{i=1}^{n} i!\,\varphi_i^2\,(\mathrm{cov}_\gamma(x,y))^i=0. \tag{3.59}$$

This equation can be solved for $\mathrm{cov}_\gamma(x,y)$ numerically for all pairs $x$ and $y$ of nodes $\mathcal{N}$ of the triangulation $\mathcal{T}_h$ of the domain $\mathcal{D}$.

**Remark 3.7.** *Solving Eq. (3.59) for all $(x,y)\in\mathcal{N}\times\mathcal{N}$ can be very time-consuming if the number of nodes is large. The efficiency of this computation can be greatly increased, if the random field is stationary and has monotone covariance function. In this case, the covariance between two points $x$ and $y$ of the target field depends only on the covariance between $x$ and $y$ of the underlying Gaussian field, i.e. $cov_r(x,y)=f(cov_\gamma(x,y))$, where $f$ does not depend on $x$ or $y$. Since the Gaussian random field has a standard normal distribution, its covariance is limited to the range $[-1,1]$, and thus it is enough to solve Eq. (3.59) on this range. To do this efficiently one can solve Eq. (3.59) at say $N=1000$ (equidistantly spaced) points in $[-1,1]$ and then use some appropriate interpolation method, e.g. monotonic cubic splines, to get very accurate approximations for $cov_\gamma(x,y)$.*

**Remark 3.8.** *It should be noted that this covariance transformation is not always possible. There are cases in which the prescribed marginal densities and the target covariance function do not match, i.e. there is no covariance function for a Gaussian field that would yield the target covariance. This is mainly due to the following two reasons: Eq. (3.56) has no solution, which is termed "bivariate level" or "Type I" incompatibility, or the resulting covariance function is not positive definite, which is called "multivariate level" or "Type II" incompatibility, see e.g. [76]. As a remedy, Phoon et al. propose a method to construct translation Gaussian fields that provide a kind of best match to the target marginal density and covariance function [76]. Since this is not vital to this work, this approach has not been pursued here further, and the input*

*random fields will be restricted to the class of translation Gaussian random fields with compatible marginal densities and covariance function.*

### 3.4.2 Transformation of the Gaussian random field

Given the covariance of the underlying Gaussian random field on $\mathcal{N} \times \mathcal{N}$, we can compute its KLE

$$\gamma(x, \omega) = \sum_{k=1}^{\infty} g_k(x)\theta_k \tag{3.60}$$

noting that the mean of $\gamma$ is zero. The $\theta_k$ are uncorrelated due to the KLE, and jointly Gaussian since they are linear transforms of the Gaussian random field $\gamma$, thus independent. The KL eigenvalues have been subsumed into the $g_k$, so only the $\theta_k$ are normalised. Together with Eq. (3.54) we get

$$r(x, \omega) = \sum_{i=0}^{\infty} \varphi_i \, H_i\Big( \sum_{k=1}^{\infty} g_k(x)\theta_k \Big). \tag{3.61}$$

Getting from there to the PCE

$$r(x, \omega) = \sum_{\alpha \in \mathcal{J}} r_\alpha(x) \, H_\alpha(\boldsymbol{\theta}), \tag{3.62}$$

first the following lemma is needed.

**Lemma 3.9.** *Let* $\boldsymbol{g} = (g_1, g_2, \ldots)^T \in \ell_2(\mathbb{N})$ *with* $\|\boldsymbol{g}\| = 1$ *and* $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots)^T \in L_2(\Omega)^{\mathbb{N}}$ *and each* $\theta_i \sim \mathcal{N}(0, 1)$. *Then*

$$H_p(\boldsymbol{g}^T\boldsymbol{\theta}) = \sum_{\alpha \in \mathcal{J}, |\alpha| = p} \binom{p}{\alpha} \boldsymbol{g}^\alpha H_\alpha(\boldsymbol{\theta}) \tag{3.63}$$

*where* $\binom{p}{\alpha}$ *is the multinomial coefficient.*

*Proof.* In the relation $\mathbb{E}\left[\partial_\alpha f\right] = \mathbb{E}\left[f H_\alpha\right]$ (see e.g. [45, 59]) set

$$f(\boldsymbol{\theta}) = H_p(\boldsymbol{g}^T\boldsymbol{\theta}). \tag{3.64}$$

Further, using $\partial H_n = nH_{n-1}$ (as the Hermite polynomials are a Scheffer sequence [79]) and simple induction on $\alpha$ we get

$$\partial_\alpha f = \partial_\alpha H_p(\boldsymbol{g}^T\boldsymbol{\theta}) \tag{3.65}$$

$$= \frac{p!}{(p-|\alpha|)!}\boldsymbol{g}^\alpha H_{p-|\alpha|}(\boldsymbol{g}^T\boldsymbol{\theta}). \tag{3.66}$$

Since $\|\boldsymbol{g}\| = 1$ we know $\boldsymbol{g}^T\boldsymbol{\theta} \sim \mathcal{N}(0,1)$ and thus $\mathbb{E}\left[H_n(\boldsymbol{g}^T\boldsymbol{\theta})\right] = \delta_{n,0}$ for any $n \geq 0$. Using this we see that $\mathbb{E}\left[fH_\alpha\right] = \mathbb{E}\left[\partial_\alpha f\right] = p!\,\boldsymbol{g}^\alpha\delta_{|\alpha|,p}$. Inserting this into the expansion

$$f(\boldsymbol{\theta}) = \sum_{\alpha\in\mathcal{J}} \mathbb{E}\left[fH_\alpha\right]/\alpha!\, H_\alpha(\boldsymbol{\theta}) \tag{3.67}$$

gives Eq. (3.63). □

**Theorem 3.10.** *Let $r$ and $\gamma$ be random fields as stated above and $\varphi$ a transformation mapping $\gamma$ to $r$. Then the coefficients $r_\alpha(x)$ of the PCE of $r$ can be obtained through the Hermite coefficients $\varphi_p$ of $\varphi$ and the KL eigenfunction $g_k(x)$ of $\gamma$ via*

$$r_\alpha(x) = \binom{|\alpha|}{\alpha}\varphi_{|\alpha|}\boldsymbol{g}(x)^\alpha \tag{3.68}$$

*where $\binom{|\alpha|}{\alpha} = \frac{(\alpha_1+\alpha_2+\cdots+\alpha_N)!}{\alpha_1!\alpha_2!\cdots\alpha_N!}$ is the multinomial coefficient and $\boldsymbol{g} = (g_1, g_2, \ldots)$.*

*Proof.* Setting the polynomial chaos expansion of $r$ equal to the expansion of $\varphi$ and inserting there the expansion of $\gamma$ gives

$$\sum_{\alpha\in\mathcal{I}} r_\alpha(x)H_\alpha(\boldsymbol{\theta}) = \sum_{p=0}^{\infty}\varphi_p H_p\Big(\sum_{i=1}^{\infty} g_i(x)\theta_i\Big). \tag{3.69}$$

Since $\gamma(x,\cdot) = \sum_{k=1}^{\infty} g_k(x)\theta_k \sim \mathcal{N}(0,1)$ at each $x \in \mathcal{D}$, it follows that $\|\boldsymbol{g}(x)\|_2^2 = \sum_{k=1}^{\infty} g_k^2(x) = 1$. Thus Lemma 3.9 can be applied on the right hand side, and projecting both sides onto $H_\beta(\boldsymbol{\theta})$ then

leads to

$$r^{(\beta)}\beta! = \mathbb{E}\left[\sum_{p=0}^{\infty}\varphi_p H_p\Big(\sum_{i=1}^{\infty} g_i(x)\theta_i\Big) H_\beta(\boldsymbol{\theta})\right] \tag{3.70}$$

$$= \mathbb{E}\left[\sum_{p=0}^{\infty}\varphi_p\left(\sum_{\alpha\in\mathcal{I},|\alpha|=p}\binom{p}{\alpha}\boldsymbol{g}(x)^\alpha H_\alpha(\boldsymbol{\theta})\right) H_\beta(\boldsymbol{\theta})\right] \tag{3.71}$$

$$= \sum_{p=0}^{\infty}\sum_{\alpha\in\mathcal{I},|\alpha|=p}\varphi_p\binom{p}{\alpha}\boldsymbol{g}(x)^\alpha\beta!\,\delta_{\alpha,\beta} \tag{3.72}$$

$$= \varphi_{|\beta|}\binom{|\beta|}{\beta}\boldsymbol{g}(x)^\beta\beta!\,, \tag{3.73}$$

concluding the proof. □

**Remark 3.11.** *In the following, it shall be briefly described how a random field $r$ is discretised in the numerical codes developed for this thesis. Given a target covariance function $cov_r$ together with some probability distribution and its corresponding transform function $\varphi$, Eq. (3.59) is solved to approximate the covariance function $cov_\gamma$ of the underlying Gaussian random field. The KLE of the Gaussian field is then computed from $cov_\gamma$ and truncated after $m_r$ terms. Given some polynomial degree $p_r$, the PCE of the field $r$ for the multiindex set $\mathcal{I}_{m_r,p_r}$ is computed using Eq. (3.68). In many cases a KLE of this field is then computed using the methods described in Section 3.3.2 and truncated after $l_r$ terms. The complete algorithm thus needs the five tuple $(\varphi, cov_r, m_r, p_r, l_r)$ as inputs. For a sample implementation of the complete algorithm see* `expand_field_kl_pce` *and* `expand_field_pce_sg` *in* sglib. *The code for the transformation of the covariance function and for the computation of the PCE coefficients can be found in* `transform_covariance_pce` *and* `pce_transform_multi`, *respectively.*

# Chapter 4

# Numerics of Tensor Products

*Algebraic properties of tensor products were discussed in Chapter 2. In order to use tensor quantities in numerical algorithms, additional issues and questions have to be resolved:*

- *How can tensors be represented in a memory efficient way?*
- *Is the approximation problem well defined for those representations and are there efficient algorithms to solve it?*
- *How can vector space operations and arithmetic be performed on those representation?*
- *Are there efficient algorithms for those operations and what is their complexity?*

*These questions have been addressed in recent years under the heading of numerical multilinear algebra. For an overview, including the historical roots of the field, see the report by Kolda and Bader [52]. For a review on more recent advances, see the article by Khoromskij [48]. Software packages that implement tensor formats and arithmetic are also available, most notable the* TensorToolbox *by Bader and Kolda [7] and the* N-way toolbox *by Andersson and Bro [2].*

*Most of the recent literature in numerical multilinear algebra directly focuses on the numerical representation of tensors in finite dimensions, viewing tensors basically as multidimensional arrays of numbers and all other representations as approximations of those objects. This thesis will pursue an approach in which the representations and methods ensue from the algebraic properties of the tensor product as outlined in Chapter 2. Tensors*

*themselves are regarded as abstract entities and multidimensional arrays as just one way of representation, which is equivalent via isomorphisms to other tensor formats.*

*The exposition will first focus on second order tensors, where efficient methods for tensor approximation are available via the singular value decomposition (SVD), and later move on to higher order tensors, where some properties and methods simply carry over, while others become substantially more complicated or have no efficient solution as yet.*

## 4.1 Second order tensors

Most of the following can be found in review articles on numerical multilinear algebra, e.g. the one by Bader and Kolda [51, 52]. However, the notation used here is different and the presentation is mainly restricted to second order tensors.

For tensors in $\mathbb{R}^N \otimes \mathbb{R}^M$ there are two commonly used representations, which arise both from the two general constructions for tensor product spaces described in Section 2.1. The first representation derives from the construction using a tensor product basis and represents tensors as matrices or long vectors, i.e. as elements of $\mathbb{R}^{N\times M}$ or $\mathbb{R}^{NM}$, respectively. While objects in this format are in some publications equated with *tensors* (see e.g. [51]), they will be denoted here as *full tensors* to avoid confusion with elements of abstract tensor product spaces that have no specified representation. The second representation is based on the construction using formal sums of pairs and represents the tensors as a sum of rank one outer products. This is often called the *canonical format* or *canonical representation* (see e.g. [51]).

To make the types of representation clearly distinct, different typefaces were chosen for each. In the following a bold, upright symbol like $\mathbf{x}$ denotes a general tensor, without concrete, specified representation. This can be seen analogously to the case of vectors in linear algebra, which can be defined and used without reference to a basis, and which only gain coordinate values when a basis is specified. In the same way, a tensor $\mathbf{x}$ can be viewed as an

abstract quantity that only takes on a specific form and specific numerical values when a representation is given. To separate the representation from the abstract object itself, different notation will be employed.

When a tensor is represented in matrix format a bold, italic, uppercase symbol, like e.g. $\boldsymbol{X}$, is used in this thesis. So, the expression $\mathbf{x} := \boldsymbol{X}$ denotes that a matrix format has been chosen as the concrete representation of the abstract tensor $\mathbf{x}$. For tensors represented in vector format bold, italic, lowercase symbols are used, which is denoted e.g. by $\mathbf{x} := \boldsymbol{x}$. The representation in the canonical format is denoted by the use of bold, upright, sans serif symbols such as $\mathbf{x} := \mathbf{x}$ (see Section 4.1.4).

Numerical algorithms for solving linear systems can often be expressed without resorting to the actual representation of vectors and linear operators. Everything that is required is that all operations for normed vectors spaces, i.e. addition, multiplication with scalars, and computation of norms (and sometimes inner products), are defined. Further, the application of linear operators needs to be compatible with the vector representation. In order to use tensor formats in linear solvers, it is therefore necessary to derive the concrete operations from the properties and requirements that were defined abstractly for any tensor representation.

In the following, conforming to the notation of Chapter 2, a concrete vector space representing the tensor product of the vectors spaces $\mathbb{R}^N$ and $\mathbb{R}^M$ will be denoted by $\mathcal{T}$, and the universal bilinear map from $\mathbb{R}^N \times \mathbb{R}^M$ into $\mathcal{T}$ by $\tau$. The equivalence relations Eq. (2.1)–(2.3) then turn into relations the vector space operations on $\mathcal{T}$ have to fulfil. Addition on $\mathcal{T}$ needs to satisfy

$$\tau(\boldsymbol{x}_1 + \boldsymbol{x}_2, \boldsymbol{y}) = \tau(\boldsymbol{x}_1, \boldsymbol{y}) + \tau(\boldsymbol{x}_2, \boldsymbol{y}) \tag{4.1}$$

$$\tau(\boldsymbol{x}, \boldsymbol{y}_1 + \boldsymbol{y}_2) = \tau(\boldsymbol{x}, \boldsymbol{y}_1) + \tau(\boldsymbol{x}, \boldsymbol{y}_2), \tag{4.2}$$

and scalar multiplication

$$\tau(\boldsymbol{x}, \alpha\boldsymbol{y}) = \alpha\tau(\boldsymbol{x}, \boldsymbol{y}) = \tau(\alpha\boldsymbol{x}, \boldsymbol{y}). \tag{4.3}$$

In general, the computation of norms when the tensors are

considered as vectors may depend on the chosen basis. Since this is undesirable for tensor representations, norms in this work are required to be unitarily invariant. Of the unitarily invariant norms the 2-norm (or Frobenius norm) will be preferred, since it is the only one that arises from a scalar product and can be computed efficiently in any of the treated representations.

In Section 2.3 it was shown that for the map $\lambda$, which maps tensor products of linear operators into linear operators on the tensor product space, the relation

$$\lambda(\boldsymbol{A} \otimes \boldsymbol{B})(\tau(\boldsymbol{x}, \boldsymbol{y})) = \tau(\boldsymbol{A}\boldsymbol{x}, \boldsymbol{B}\boldsymbol{y}) \tag{4.4}$$

must hold. Because the way that the operator $\lambda(\boldsymbol{A} \otimes \boldsymbol{B})$ acts on the tensor $\tau(\boldsymbol{x}, \boldsymbol{y})$ is completely specified by the tensor format $\tau$, Eq. (4.4) can be used as the defining relation for $\lambda$ for each such format. Since a linear operator is completely specified by its action on a basis of its domain and the elementary tensors include a basis of the tensor product space, the mapping $\lambda$ can be extended naturally, i.e. by linearity, to the full tensor product space.

### 4.1.1 Representations

In the following the preceding abstract requirements for tensor arithmetic are made concrete for the different tensor representations. The presentation starts with full tensor formats, namely the matrix and the vector format, which is equivalent to the notion of the Kronecker product, and concludes with a low-rank format, the so-called canonical representation.

### 4.1.2 Matrix format

The most common way to represent elements of $\mathbb{R}^N \otimes \mathbb{R}^M$ is as matrices, i.e. as elements of $\mathcal{T} = \mathbb{R}^{N \times M}$. The map $\tau : \mathbb{R}^N \times \mathbb{R}^M \to \mathcal{T}$ can be defined as $\tau(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}\boldsymbol{y}^T$, or speaking in terms of representation $\boldsymbol{x} \otimes \boldsymbol{y} := \boldsymbol{x}\boldsymbol{y}^T$.

### Addition and scalar multiplication

Addition of tensors in matrix representation is elementwise, i.e. $[\boldsymbol{X}+\boldsymbol{Y}]_{ij} = [\boldsymbol{X}]_{ij} + [\boldsymbol{Y}]_{ij}$. While this may be obvious, it can also be formally derived from $\boldsymbol{X} = \sum_{i,j} x_{ij}\boldsymbol{e}_i\boldsymbol{e}_j^T = \sum_{i,j} x_{ij}\tau(\boldsymbol{e}_i, \boldsymbol{e}_j)$ and the bilinearity of $\tau$. The operation count for addition is thus $\Theta(NM)$[1].

Multiplication with scalars in matrix representation is again elementwise, i.e. $[\alpha\boldsymbol{X}]_{ij} = \alpha[\boldsymbol{X}]_{ij}$, as the same reasoning as for addition applies here also. The operation count for scalar multiplication is thus $\Theta(NM)$.

### Norms and inner products

Note first that even when the a tensor is represented in matrix format the norms of interest here are *vector norms*. I.e. the $p$-norm of a tensor $\mathbf{x}$ should be independent of representation, so for the matrix format the $p$-norm is defined by

$$\|\boldsymbol{X}\|_p = \left( \sum_{i=1}^{N} \sum_{j=1}^{M} |[\boldsymbol{X}]_{ij}|^p \right)^{1/p} \tag{4.5}$$

since $\boldsymbol{X}$ is essentially a vector in this context, not a linear map.

**Remark 4.1.** *Note that for the matrix norm induced by the vector $p$-norm the notation $|||\boldsymbol{X}|||_p$ is used in this thesis instead of $\|\boldsymbol{X}\|_p$, which is purely reserved for vector norms. Some care has to be taken in implementations that indeed the vector $p$-norms are computed in this case. In Matlab® e.g.* `norm(X(:),p)` *must be used to actually compute the vector $p$-norm, instead of the induced matrix $p$-norm, which would be computed by* `norm(X,p)`*.*

The norm that will be mainly used is the *vector* 2-norm, as this is the only $p$-norm that can be efficiently computed in other

[1] For comparison of runtime and memory requirements of tensor formats and algorithms the symbol $\Theta$ is used throughout this thesis for the exact asymptotic order in Landau notation. The symbol O, which is in more widespread use, denotes an asymptotic *upper bound* and is unsuitable for meaningful comparisons.

tensor representations. In the matrix case the vector 2-norm also corresponds to the Frobenius norm, i.e. $\|\boldsymbol{X}\|_2 = \|\|\boldsymbol{X}\|\|_F$. The inner product between tensors corresponds in this representation to the Frobenius inner product between matrices

$$\langle \mathbf{x}|\mathbf{y}\rangle = \langle \boldsymbol{X}|\boldsymbol{Y}\rangle_F = \mathrm{Tr}(\boldsymbol{X}\boldsymbol{Y}^T) = \sum_{i=1}^{M}\sum_{j=1}^{N}[\boldsymbol{X}]_{ij}[\boldsymbol{Y}]_{ij}. \tag{4.6}$$

In practice, of course the expression containing the trace is not used, but rather the last expression containing the double sum. If $\odot$ denotes the (elementwise) Hadamard product, and the $\Sigma$ function denotes summing over all matrix elements, this can be written succinctly as

$$\langle \mathbf{x}|\mathbf{y}\rangle = \Sigma(\boldsymbol{X} \odot \boldsymbol{Y}). \tag{4.7}$$

The operation count for the computation of the inner product and therefore also for the norm is $\Theta(NM)$.

### Operator application

As outlined in Section 4.1, the application of a tensor operator $\boldsymbol{A} \otimes \boldsymbol{B}$ needs to be consistent with the chosen tensor representation, i.e. $\lambda(\boldsymbol{A} \otimes \boldsymbol{B})\tau(\boldsymbol{x}, \boldsymbol{y}) = \tau(\boldsymbol{A}\boldsymbol{x}, \boldsymbol{B}\boldsymbol{y})$. So for simple tensors in matrix representation we have

$$\lambda(\boldsymbol{A} \otimes \boldsymbol{B})(\boldsymbol{x}\boldsymbol{y}^T) = (\boldsymbol{A}\boldsymbol{x})(\boldsymbol{B}\boldsymbol{y})^T \tag{4.8}$$

$$= \boldsymbol{A}(\boldsymbol{x}\boldsymbol{y}^T)\boldsymbol{B}^T \tag{4.9}$$

Since this must hold for all basis vectors of $\mathbb{R}^N$ and $\mathbb{R}^M$, it follows from extension by linearity that

$$\lambda(\boldsymbol{A} \otimes \boldsymbol{B})(\boldsymbol{X}) = \boldsymbol{A}\boldsymbol{X}\boldsymbol{B}^T \tag{4.10}$$

holds for any matrix representation $\boldsymbol{X}$ of the tensor $\mathbf{x}$. The runtime for this operation depends on the cost of the two matrix-matrix multiplications. As the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are in general sparse matrices it is not appropriate to estimate the complexity

of these operations purely in terms of the size of the matrices, because the runtime depends highly on the sparsity structure of the matrices and the data format used. The runtime will therefore be described by $\tau_A$ and $\tau_B$ as time used per matrix application to one column vector. The total runtime for (4.10) is thus

$$\tau_A \Theta(N) + \tau_B \Theta(M) \tag{4.11}$$

as matrix $\boldsymbol{A}$ has to be applied to $N$ columns and $\boldsymbol{B}^T$ to $M$ rows.

### 4.1.3 The Kronecker product

The Kronecker product is a product between matrices that is often regarded as *the* tensor product between matrices (see e.g. [41]). For vectors as special matrices with only one column it is a tensor product of vector spaces, and for matrices it is a tensor product of linear operators, which is compatible with that for vectors. For a summary of many interesting relations and applications of the Kronecker product see the article by Van Loan [87].

The Kronecker product between two matrices $\boldsymbol{A} \in \mathbb{R}^{P \times M}$ and $\boldsymbol{B} \in \mathbb{R}^{Q \times N}$ is defined as

$$\boldsymbol{A} \hat{\otimes} \boldsymbol{B} = \begin{bmatrix} a_{11}\boldsymbol{B} & \cdots & a_{1M}\boldsymbol{B} \\ \vdots & \ddots & \vdots \\ a_{P1}\boldsymbol{B} & \cdots & a_{PM}\boldsymbol{B} \end{bmatrix}, \tag{4.12}$$

which is a matrix of size $PQ \times MN$. Note that the Kronecker product is denoted by $\hat{\otimes}$ here, to distinguish it from the abstract tensor product denoted by $\otimes$. For vectors $\boldsymbol{x} \in \mathbb{R}^M$ and $\boldsymbol{y} \in \mathbb{R}^N$ applying the same definition yields

$$\boldsymbol{x} \hat{\otimes} \boldsymbol{y} = \begin{bmatrix} x_1\boldsymbol{y} \\ \vdots \\ x_M\boldsymbol{y} \end{bmatrix}. \tag{4.13}$$

Note that the Kronecker product of two vectors is not a matrix but a long thin vector. It is obvious that the Kronecker product

for vectors in Eq. (4.13) defines a tensor product as the right hand side is trivially isomorphic to $\boldsymbol{x}\boldsymbol{y}^T$ via the matrix stacking operation, i.e. $\boldsymbol{x} \hat{\otimes} \boldsymbol{y} = \mathrm{vec}((\boldsymbol{x}\boldsymbol{y}^T)^T)$. The Kronecker product can be seen just as a reordering of the matrix tensor format by mapping the indices by $(i, j) \mapsto j + (i-1)N$, which is an invertible map from $[1, M] \times [1, N] \to [1, NM]$. As this defines an isomorphism between $R^{M \times N}$ and $R^{MN}$ the Kronecker product defines a tensor product via $\tau(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x} \hat{\otimes} \boldsymbol{y}$ into $\mathcal{T} = \mathbb{R}^{MN}$.

Addition and scalar multiplication are the standard elementwise operations on vectors. Memory and runtime for those operations is as in the matrix format of the order $\Theta(NM)$.

It can be shown that the Kronecker product satisfies the equality

$$(\boldsymbol{A} \hat{\otimes} \boldsymbol{B})(\boldsymbol{x} \hat{\otimes} \boldsymbol{y}) = (\boldsymbol{A}\boldsymbol{x}) \hat{\otimes} (\boldsymbol{B}\boldsymbol{y}). \tag{4.14}$$

Together with the requirement Eq. (4.4) for the mapping $\lambda$ for the Kronecker product, i.e. $\lambda(\boldsymbol{A} \otimes \boldsymbol{B})(\boldsymbol{x} \hat{\otimes} \boldsymbol{y}) = (\boldsymbol{A}\boldsymbol{x}) \hat{\otimes} (\boldsymbol{B}\boldsymbol{y})$, it follows that $\lambda$ can be represented by

$$\lambda(\boldsymbol{A} \otimes \boldsymbol{B}) = \boldsymbol{A} \hat{\otimes} \boldsymbol{B}. \tag{4.15}$$

This means that the same product can be used for the representation of the tensor product of linear operators as for the tensor product of vectors.

**Remark 4.2.** *Note also that it is sometimes customary to denote by Kronecker product representations products that are in tensor form, which could be interpreted as a Kronecker product, but in which the Kronecker product is not actually carried out (see e.g. the Kronecker product preconditioner in Section 7.1.2).*

### Reverse Kronecker Product

In the following an alternate definition for the Kronecker product is proposed. Using the standard definition of the Kronecker product in computer codes, often the order of factors has to be reversed when converting between different tensor representations. Forgetting this reversal at the right points usually leads to pro-

grams, that still work, but compute wrong results, leading thus to subtle, hard to detect bugs. Therefore, first the cause of the problem will be analysed, and then an alternate definition, called the *reverse Kronecker product*, will be given, which does not suffer from this problem.

One short-coming of the standard definition of the Kronecker product is, that it is not compatible with the vectorisation operation, usually denoted by vec, which maps a matrix to a vector by stacking its columns, i.e.

$$\operatorname{vec}(\boldsymbol{A}) = \operatorname{vec}\left(\begin{bmatrix}\boldsymbol{a}_1 & \boldsymbol{a}_2 & \dots & \boldsymbol{a}_n\end{bmatrix}\right) = \begin{bmatrix}\boldsymbol{a}_1 \\ \boldsymbol{a}_2 \\ \vdots \\ \boldsymbol{a}_n\end{bmatrix}.$$

When building the outer product $\boldsymbol{x}\boldsymbol{y}^T$ of two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, we have the relation

$$\operatorname{vec}(\boldsymbol{x}\boldsymbol{y}^T) = \boldsymbol{y} \hat{\otimes} \boldsymbol{x}, \tag{4.16}$$

reversing the order of factors in the right hand side. Similar relations hold if matrices are involved, e.g.

$$(\boldsymbol{A} \hat{\otimes} \boldsymbol{B}) \operatorname{vec}(\boldsymbol{X}) = \operatorname{vec}(\boldsymbol{B}\boldsymbol{X}\boldsymbol{A}^T) \tag{4.17}$$

for matrices $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{X}$ (see [87]).

In computer codes conversion between different representations is frequently needed, e.g. is the matrix format in many respects better to handle internally, but off-the-shelve linear solvers take only thin vectors, so that conversion is necessary. The reversal of the order of the operands in the Kronecker product is then a common and hard to detect source of errors. A simple, but effective remedy for this is defining the product in the reversed way, i.e.

$$\boldsymbol{A} \check{\otimes} \boldsymbol{B} = \begin{bmatrix} b_{11}\boldsymbol{A} & \cdots & b_{1N}\boldsymbol{A} \\ \vdots & \ddots & \vdots \\ b_{Q1}\boldsymbol{A} & \cdots & b_{QN}\boldsymbol{A} \end{bmatrix} = \boldsymbol{B} \hat{\otimes} \boldsymbol{A}. \tag{4.18}$$

With the reversed Kronecker product most identities come out in a more natural way, e.g. instead of Eq. (4.17) we have

$$\mathrm{vec}(\boldsymbol{x}\boldsymbol{y}^T) = \boldsymbol{x} \mathbin{\check{\otimes}} \boldsymbol{y} \tag{4.19}$$

and

$$(\boldsymbol{A} \mathbin{\check{\otimes}} \boldsymbol{B})\,\mathrm{vec}(\boldsymbol{X}) = \mathrm{vec}(\boldsymbol{A}\boldsymbol{X}\boldsymbol{B}^T). \tag{4.20}$$

As the order of factors stays consistent this way, one source of errors can be eliminated using this definition. Hence, in computer codes developed for this work, only this definition has been used.

### 4.1.4 Low-rank format

If the matrix representation of a tensor has a rank that is substantially lower than $\min(N, M)$, there are formats in which it can be represented much more economically (see e.g. [51]). We will call every such format that uses less memory than $MN$ a "low-rank format". For tensors of order two there is effectively only one low-rank format as this can be proven to be optimal. The name in the literature for this representation is not unique, sometimes it is called separated format, canonical format or sometimes CP format (for CANDECOMP/PARAFAC)[51]. In this work the term canonical format is used for exactly this representation. If any other format with memory requirements much lower than $MN$ could be substituted, the generic term low-rank format will be employed.

As shown in Section 2.1 any tensor from a finite dimensional tensor product space $\mathcal{U} \otimes \mathcal{V}$ can be represented as a finite sum

$$\mathbf{x} = \sum_{i=1}^{R} s_i \boldsymbol{x}_i \otimes \boldsymbol{y}_i \tag{4.21}$$

for some integer $R$, and vectors $\boldsymbol{x}_i \in \mathcal{U}$ and $\boldsymbol{y}_i \in \mathcal{V}$. The integer $R$ is called the rank of $\mathbf{x}$ if it is the smallest such integer, and otherwise just the *numerical rank* (sometimes also *separation rank*). If the vectors $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ are collected into matrices $\boldsymbol{X}$ and

$\boldsymbol{Y}$ the tensor $\mathbf{x}$ can be represented as

$$\mathbf{x} := \mathbf{x} = (\boldsymbol{s}; \boldsymbol{X}, \boldsymbol{Y}) \tag{4.22}$$

where $\mathbf{x}$ denotes the representation in canonical format. The relation to the matrix format from Section 4.1.2 is given by

$$\hat{\boldsymbol{X}} = \sum_{i=1}^{R} s_i \boldsymbol{x}_i \boldsymbol{y}_i^T = \boldsymbol{X}\boldsymbol{S}\boldsymbol{Y}^T \tag{4.23}$$

where the matrix representation is now referred to by $\hat{\boldsymbol{X}}$, and

$$\boldsymbol{S} = \operatorname{diag}(\boldsymbol{s}) = \begin{bmatrix} s_1 & & \\ & \ddots & \\ & & s_R \end{bmatrix} \tag{4.24}$$

is the diagonal matrix formed by the entries of $\boldsymbol{s}$. The memory requirements for this format are $\Theta(R(N+M))$, which is substantially lower than for the matrix format with $\Theta(NM)$ if $R \ll \min(N, M)$.

The representation as defined in Eq. (4.22) is not unique. Any matrix decomposition like in Eq. (4.23) yields a valid representation in the canonical format. The preferred method, however, is to use the singular value decomposition (SVD) here. The SVD factors $\hat{\boldsymbol{X}}$ into the product

$$\hat{\boldsymbol{X}} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T \tag{4.25}$$

such that $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthogonal and $\boldsymbol{\Sigma}$ is diagonal with non-negative entries $\sigma_1 \geq \sigma_2 \geq \cdots \sigma_R > 0$ and $\sigma_{R+1} \cdots \sigma_{\min(M,N)} = 0$, called singular values. The number of non-zero singular values equals the rank of the matrix. The last equation can also be written in the form

$$\hat{\boldsymbol{X}} = \sum_{i=1}^{R} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T \tag{4.26}$$

The special significance of the SVD comes from the fact that, if Eq. (4.26) is truncated to some $R' < R$, the truncated sum yields the best approximation from all rank-$R'$ matrices in the Frobenius norm. If

$$\hat{\boldsymbol{X}}_{R'} = \sum_{i=1}^{R'} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T \tag{4.27}$$

denotes the truncated sum, the following result holds

$$\left|\left|\left|\hat{\boldsymbol{X}} - \hat{\boldsymbol{X}}_{R'}\right|\right|\right|_F = \min_{\mathrm{rank}(\boldsymbol{Z}) \leq R'} \left|\left|\left|\hat{\boldsymbol{X}} - \boldsymbol{Z}\right|\right|\right|_F. \tag{4.28}$$

This is the well-known Eckart-Young theorem (see e.g. [3]) and holds also for any other unitarily invariant norm. If we denote the (unitarily invariant) Schatten-$p$ norms by $|||\cdot|||_{S_p}$—the Frobenius norm is a special case corresponding to the Schatten-2 norm—the truncation error can be expressed as

$$\left|\left|\left|\hat{\boldsymbol{X}} - \hat{\boldsymbol{X}}_{R'}\right|\right|\right|_{S_p} = \left( \sum_{i=R'+1}^{R} \sigma_i^p \right)^{1/p}. \tag{4.29}$$

As has already been pointed out in Section 3.3.2 the SVD is equivalent to the KLE, if the $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthogonal with respect to the spatial and stochastic Gram matrices. This can be achieved either by solving from the left and right with the Cholesky factors of the Gram matrices, or by performing a modified Gram-Schmidt process first [29, 30]. Eq. (4.29) for $p = 2$ gives then directly the $L_2$ approximation error.

### Addition and scalar multiplication

Addition of tensors in low-rank format is very straightforward. Since for the general tensor product we have

$$\mathbf{x}_1 + \mathbf{x}_2 = \sum_{i=1}^{R_1} s_{1,i} \boldsymbol{x}_{1,i} \otimes \boldsymbol{y}_{1,i} + \sum_{i=1}^{R_2} s_{2,i} \boldsymbol{x}_{2,i} \otimes \boldsymbol{y}_{2,i}, \tag{4.30}$$

it follows that in the canonical representation the sum is represented as

$$\mathbf{x}_1 + \mathbf{x}_2 = \left( \begin{bmatrix} \boldsymbol{s}_1 \\ \boldsymbol{s}_2 \end{bmatrix}; \begin{bmatrix} \boldsymbol{X}_1 & \boldsymbol{X}_2 \end{bmatrix}, \begin{bmatrix} \boldsymbol{Y}_1 & \boldsymbol{Y}_2 \end{bmatrix} \right), \tag{4.31}$$

where the square brackets denote the concatenation of the matrices, i.e. $\begin{bmatrix} \boldsymbol{X}_1 \, \boldsymbol{X}_2 \end{bmatrix}$ is the block matrix formed by the columns of $\boldsymbol{X}_1$ and then the columns of $\boldsymbol{X}_2$. This operation can be performed in $\Theta(1)$ operations, if no memory has to be moved, and only pointers are manipulated. Since most systems move the memory in order to keep memory locations contiguous which is of order $\Theta(R(N+M))$, the cost for this operation will be denoted by $\mathrm{O}(R(N+M))$. Scalar multiplication is defined by

$$\alpha \mathbf{x} = (\alpha \boldsymbol{s}; \boldsymbol{X}, \boldsymbol{Y}) \tag{4.32}$$

which makes it an $\Theta(R)$ operation.

### Norms and inner products

From the definition of the inner product for tensor product Hilbert spaces (see Section 2.2), the inner product between tensors $\mathbf{x}_i := \mathbf{x}_i = (\boldsymbol{s}_i; \boldsymbol{X}_i, \boldsymbol{Y}_i)$ for $i \in \{1, 2\}$ can be expressed as

$$\langle \mathbf{x}_1 | \mathbf{x}_2 \rangle = \langle \mathbf{x}_1 | \mathbf{x}_2 \rangle = \Big\langle \sum_{i=1}^{R_1} s_{1,i} \boldsymbol{x}_{1,i} \otimes \boldsymbol{y}_{1,i} \;\Big|\; \sum_{j=1}^{R_2} s_{2,j} \boldsymbol{x}_{2,j} \otimes \boldsymbol{y}_{2,j} \Big\rangle \tag{4.33}$$

$$= \sum_{i,j=1}^{R_1,R_2} s_{1,i} s_{2,j} \langle \boldsymbol{x}_{1,i} | \boldsymbol{x}_{2,j} \rangle \langle \boldsymbol{y}_{1,i} | \boldsymbol{y}_{2,j} \rangle \tag{4.34}$$

This can also be written (and efficiently evaluated numerically) as

$$\langle \mathbf{x}_1 | \mathbf{x}_2 \rangle = \Sigma((\boldsymbol{s}_1 \boldsymbol{s}_2^T) \odot (\boldsymbol{X}_1^T \boldsymbol{X}_2) \odot (\boldsymbol{Y}_1^T \boldsymbol{Y}_2)), \tag{4.35}$$

using the notation introduced in Section 4.1.2. The operation count for this computation comes from the matrix products and

from the Hadamard product. Since the matrices multiplied by the Hadamard product have size $R_1 \times R_2$, the operation count for this part is $\Theta(R_1 R_2)$. The operation counts for the outer products are $\Theta(R_1 R_2)$, $\Theta(M R_1 R_2)$ and $\Theta(N R_1 R_2)$ respectively so that the total operation count is $\Theta((N+M)R_1 R_2)$.

The norm of a tensor $\mathbf{x}$ can be computed via the scalar product by

$$\|\mathbf{x}\| = \langle \mathbf{x}|\mathbf{x}\rangle^{1/2}. \tag{4.36}$$

Note, however that this may suffer severely from cancellation if this formula is applied directly to the difference between two tensors like one has to do for the computation of errors. Especially, if $\mathbf{x}$ and $\tilde{\mathbf{x}}$ are close, the result of computing $\langle \mathbf{x}-\tilde{\mathbf{x}}|\mathbf{x}-\tilde{\mathbf{x}}\rangle$ may be orders of magnitude off the mark, such that e.g. convergence of iterative methods cannot be observed. To alleviate that problem one can either bring $\mathbf{x}-\tilde{\mathbf{x}}$ into a normalised form, or use the algorithms as described in Appendix B. If the representation has been normalised e.g. by using the truncation algorithm (see Section 4.1.5) the norm of the $\mathbf{x}$ can be computed from the singular values alone

$$\|\mathbf{x}\| = \left(\sum_{i=1}^{R} \sigma_i^2\right)^{1/2} \tag{4.37}$$

As $\mathbf{x}$ is often brought into normalised form (i.e. $\mathbf{x} = (\boldsymbol{\sigma}; \boldsymbol{X}, \boldsymbol{Y})$ with $\boldsymbol{X}$ and $\boldsymbol{Y}$ having orthonormal columns) in many of the algorithms this formula can often be used and is preferred, as it is more efficient being of order $\Theta(R)$ and also more stable than using Eq. (4.36).

### Operator application

Application of operators in the canonical format is very straightforward by

$$\lambda(\boldsymbol{A}\otimes\boldsymbol{B})(\boldsymbol{s}; \boldsymbol{X}, \boldsymbol{Y}) = (\boldsymbol{s}; \boldsymbol{A}\boldsymbol{X}, \boldsymbol{B}\boldsymbol{Y}), \tag{4.38}$$

which follows directly from the requirement Eq. (4.4). The runtime complexity is of order $\tau_A\Theta(R)+\tau_B\Theta(R) = \Theta(R) = (\tau_A+\tau_B)\Theta(R)$,

since each operator is applied only to $R$ columns.

Problematic about operator application is, however, that the operators of interest are generally *sums* of tensor products of operators. Application of such an operator sum to a tensor $\mathsf{x} = (\boldsymbol{s}; \boldsymbol{X}, \boldsymbol{Y})$ of rank $R$ in canonical format leads to

$$\sum_{i=1}^{L} \lambda(\boldsymbol{A}_i \otimes \boldsymbol{B}_i)(\boldsymbol{s}; \boldsymbol{X}, \boldsymbol{Y}) = \left( \begin{bmatrix} \boldsymbol{s} \\ \vdots \\ \boldsymbol{s} \end{bmatrix}; \left[\boldsymbol{A}_1\boldsymbol{X}, \ldots, \boldsymbol{A}_L\boldsymbol{X}\right], \left[\boldsymbol{B}_1\boldsymbol{Y}, \ldots, \boldsymbol{B}_L\boldsymbol{Y}\right] \right).$$

The numerical rank of the tensor on the right hand side is now $LR$, i.e. a factor of $L$ larger than before. It is evident that proceeding in this way in iterative methods would quickly produce tensors with huge ranks, such that no gain in memory or computational efficiency could be expected from the use of the format.

### 4.1.5 Truncation

One solution for this is to use approximate representations. Frequently, e.g. after each addition of tensors, the canonical tensor $\mathsf{x}$ with numerical rank $R$ is approximated by a tensor $\mathsf{x}'$ of smaller rank $R'$ such that the difference $\|\mathsf{x}' - \mathsf{x}\|$ is small. The effects on iterative methods that is caused by of this loss in accuracy is discussed in the next chapter. Here we shall first describe a method for efficient truncation of second order tensors in canonical format. The algorithm itself, albeit without derivation, can be found in [34].

#### The basic truncation algorithm

Suppose the tensor at hand is given in a canonical format $\mathsf{x} = (\boldsymbol{s}; \boldsymbol{X}, \boldsymbol{Y})$, such that the respective matrix representation would be

$$\boldsymbol{A} = \boldsymbol{X}\boldsymbol{S}\boldsymbol{Y}^T \tag{4.39}$$

where $\boldsymbol{A}$ is an $M \times N$ matrix, $\boldsymbol{X}$ is $M \times R$, $\boldsymbol{S} = \mathrm{diag}(\boldsymbol{s})$ is $R \times R$, and $\boldsymbol{Y}$ is $N \times R$. The question is now, how to reduce this representation to a smaller rank $R'$ such that the matrix representation stays unchanged or that the difference is minimal in the Frobenius norm for the given rank $R'$. As already discussed, this can be achieved via the SVD. The essential problem is now, how to compute the SVD of $\boldsymbol{X}\boldsymbol{S}\boldsymbol{Y}^T$ without forming the matrix $\boldsymbol{A}$ explicitly, as this would incur a huge amount of memory and runtime, that we tried to avoid in the first place.

One method to achieve this can be found in [34]. Take orthogonal decompositions of $\boldsymbol{X}$ and $\boldsymbol{Y}$

$$\begin{aligned} \boldsymbol{Q}_X \boldsymbol{R}_X &= \boldsymbol{X} \\ \boldsymbol{Q}_Y \boldsymbol{R}_Y &= \boldsymbol{Y}. \end{aligned} \tag{4.40}$$

A common choice is the QR decomposition, but also e.g. the QL, SVD, or polar decomposition could be used. Essential is only that a "thin" decomposition is used such that $\boldsymbol{R}_X$ and $\boldsymbol{R}_Y$ are $R \times R$ matrices. The cost for this type of decomposition is typically $\Theta((N+M)R^2)$, for both decompositions, only the constant factor depending on the concrete decomposition and algorithm used. Inserting Eq. (4.40) into Eq. (4.39) yields

$$\begin{aligned} \boldsymbol{A} &= (\boldsymbol{Q}_X \boldsymbol{R}_X)\boldsymbol{S}(\boldsymbol{Q}_Y \boldsymbol{R}_Y)^T \\ &= \boldsymbol{Q}_X (\boldsymbol{R}_X \boldsymbol{S} \boldsymbol{R}_Y^T) \boldsymbol{Q}_Y^T \end{aligned} \tag{4.41}$$

The right hand side of Eq. (4.41) is already similar to the SVD except that the inner term $(\boldsymbol{R}_X \boldsymbol{S} \boldsymbol{R}_Y^T)$ is in general neither diagonal nor positive definite. As this matrix is only of size $R \times R$ taking the SVD

$$\boldsymbol{U}_R \boldsymbol{\Sigma}_R \boldsymbol{V}_R^T = \boldsymbol{R}_X \boldsymbol{S} \boldsymbol{R}_Y^T \tag{4.42}$$

is a relatively cheap operation of order $\Theta(R^3)$. Inserting this factorisation into Eq. (4.41) and reordering yields

$$\boldsymbol{A} = \boldsymbol{Q}_X (\boldsymbol{U}_R \boldsymbol{\Sigma}_R \boldsymbol{V}_R^T) \boldsymbol{Q}_Y^T$$

$$= (\boldsymbol{Q}_X \boldsymbol{U}_R)\boldsymbol{\Sigma}_R(\boldsymbol{Q}_Y \boldsymbol{V}_R)^T, \tag{4.43}$$

which evidently is the SVD of $\boldsymbol{A}$ as the outer factors are orthogonal and the inner factor is diagonal and positive semi-definite.

The total cost of the preceding operations is $\Theta((N+M)R^2+R^3)$. If the QR decomposition is used for the orthogonal factorisation the number of operations (multiplications and additions) can be exactly stated as $23R^3 + 6R^2(M + N)$ [34]. If only a rank-$R'$ approximation to $\boldsymbol{A}$ is needed, only the first $R'$ singular values of $\boldsymbol{\Sigma}_R = \operatorname{diag}(\boldsymbol{\sigma}_r)$, and the first $R'$ columns of $\boldsymbol{U}_R$ and $\boldsymbol{V}_R$ can be taken in Eq. (4.43) instead of the full matrices. Summarising the steps taken in the preceding derivation leads to Alg. 4.1 (see also [34]).

**Algorithm 4.1** Fixed rank truncation

**Input:** vector $\boldsymbol{s}$, matrices $\boldsymbol{X}$, $\boldsymbol{Y}$, an integer $R'$
**Output:** vector $\boldsymbol{\sigma}$, matrices $\boldsymbol{U}$, $\boldsymbol{V}$
1: $(\boldsymbol{Q}_X, \boldsymbol{R}_X) \leftarrow \mathrm{QR}(\boldsymbol{X})$ ▷ economy QR
2: $(\boldsymbol{Q}_Y, \boldsymbol{R}_Y) \leftarrow \mathrm{QR}(\boldsymbol{Y})$
3: $(\boldsymbol{\sigma}_R, \boldsymbol{U}_R, \boldsymbol{V}_R) \leftarrow \mathrm{SVD}(\boldsymbol{R}_X \operatorname{diag}(\boldsymbol{s}) \boldsymbol{R}_Y^T)$
4: $\boldsymbol{\sigma} \leftarrow \boldsymbol{\sigma}_R(1{:}R')$
5: $\boldsymbol{U} \leftarrow \boldsymbol{Q}_X \boldsymbol{U}_R(1{:}M, 1{:}R')$
6: $\boldsymbol{V} \leftarrow \boldsymbol{Q}_Y \boldsymbol{V}_R(1{:}N, 1{:}R')$

**Remark 4.3.** *The algorithm does not necessarily need to use the QR factorisation. Important is only that the first factor in the decomposition is orthogonal—so e.g. the polar decomposition could as well be used here. In fact, to speed up computational time we don't always factor into a right triangular matrix (see Section 4.1.5). In principle one could determine* any *orthogonal basis $\boldsymbol{Q}$ of the column span of $\boldsymbol{A}$ and use $\boldsymbol{R} = \boldsymbol{Q}^T\boldsymbol{A}$. It is essential, however, that always a "thin" or "economy" decomposition is used, such that $\boldsymbol{Q}$ has the same shape as $\boldsymbol{A}$.*

The result of the computation of Alg. 4.1 can be seen as a map from the full tensor space $\mathbb{R}^N \otimes \mathbb{R}^M$ into the submanifold of

rank-$R'$ tensors. This map will be referred to as the *truncation operator* and denoted by $T_{R'}$.

In many algorithms it is better to bound the error incurred by the truncation, instead of truncating to a fixed rank $R'$. As the error in the Schatten-$p$ norm for a rank $R'$ truncation is given by

$$\left|\left|\left|\boldsymbol{A} - T_{R'}\boldsymbol{A}\right|\right|\right|_{S_p} = \left(\sum_{i=R'+1}^{R} \sigma_i^p\right)^{1/p} \tag{4.44}$$

the minimum rank $R'$ such that the error does not exceed the bound $\varepsilon$ is given by

$$R' = \operatorname*{arg\,min}_{r \leq R} \left(\sum_{i=r+1}^{R} \sigma_i^p \leq \varepsilon^p\right). \tag{4.45}$$

The modified algorithm using this formula is given by Alg. 4.2. The map implemented by this algorithm will be denoted by $T_\varepsilon$ and satisfies the inequality

$$\left|\left|\left|\boldsymbol{A} - T_\varepsilon\boldsymbol{A}\right|\right|\right|_{S_p} \leq \varepsilon. \tag{4.46}$$

The effect these truncations have, when they are applied inside iterative methods, will be studied in Section 5.1. A sample implementation of this algorithm can be found in sglib in the file `tensor/private/tensor_truncate_svd`.

### Optimised truncation algorithm

The truncation algorithm (Alg. 4.1) is chiefly run immediately after adding two tensors $\mathsf{x}_1$ and $\mathsf{x}_2$ on their sum $\mathsf{x} = \mathsf{x}_1 + \mathsf{x}_2 = (\boldsymbol{s}; \boldsymbol{X}, \boldsymbol{Y})$ where $\boldsymbol{X} = \begin{bmatrix}\boldsymbol{X}_1 & \boldsymbol{X}_2\end{bmatrix}$ and $\boldsymbol{Y} = \begin{bmatrix}\boldsymbol{Y}_1 & \boldsymbol{Y}_2\end{bmatrix}$. The algorithm needs to apply two QR decompositions both on $\boldsymbol{X}$ and on $\boldsymbol{Y}$. Very frequently, however, $\boldsymbol{X}_1$ and $\boldsymbol{Y}_1$ already have orthogonal columns due to previous truncations. In that case it is not necessary to orthogonalise the columns again, and computational time can be saved by making use of this fact. The basic idea for

**Algorithm 4.2** Epsilon rank truncation

**Input:** vector $\boldsymbol{s}$, matrices $\boldsymbol{X}$, $\boldsymbol{Y}$, a real number $\varepsilon > 0$
**Output:** vector $\boldsymbol{\sigma}$, matrices $\boldsymbol{U}$, $\boldsymbol{V}$

1: $(\boldsymbol{Q}_X, \boldsymbol{R}_X) \leftarrow \mathrm{QR}(\boldsymbol{X})$ ▷ economy QR
2: $(\boldsymbol{Q}_Y, \boldsymbol{R}_Y) \leftarrow \mathrm{QR}(\boldsymbol{Y})$
3: $(\boldsymbol{\sigma}_R, \boldsymbol{U}_R, \boldsymbol{V}_R) \leftarrow \mathrm{SVD}(\boldsymbol{R}_X \,\mathrm{diag}(\boldsymbol{s}) \boldsymbol{R}_Y^T)$
4: $R' \leftarrow \arg\min_{r \leq R} \left( \sum_{i=r+1}^{R} \sigma_i^p \leq \varepsilon^p \right)$
5: $\boldsymbol{\sigma} \leftarrow \boldsymbol{\sigma}_R(1{:}R')$
6: $\boldsymbol{U} \leftarrow \boldsymbol{Q}_X \boldsymbol{U}_R(1{:}M, 1{:}R')$
7: $\boldsymbol{V} \leftarrow \boldsymbol{Q}_Y \boldsymbol{V}_R(1{:}N, 1{:}R')$

the updating algorithm can be found in a paper by Brand [14], the improved algorithm (Alg. 4.4) was developed by the author.

As the update of the factorisation is the same for $\boldsymbol{X}$ as for $\boldsymbol{Y}$ the discussion will be restricted to the former. The orthogonal decomposition of $\boldsymbol{X} = \begin{bmatrix} \boldsymbol{X}_1 & \boldsymbol{X}_2 \end{bmatrix}$ can be written as

$$\begin{bmatrix} \boldsymbol{X}_1 & \boldsymbol{X}_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{Q}_1 & \boldsymbol{Q}_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & \boldsymbol{A} \\ \boldsymbol{0} & \boldsymbol{B} \end{bmatrix} \tag{4.47}$$

where by assumption that $\boldsymbol{X}_1$ is already orthogonal $\boldsymbol{Q}_1 = \boldsymbol{X}_1$ and the matrices $\boldsymbol{Q}_2$, $\boldsymbol{A}$ and $\boldsymbol{B}$ still have to be determined. Forming the product on the right hand side leads to

$$\boldsymbol{X}_2 = \boldsymbol{Q}_1 \boldsymbol{A} + \boldsymbol{Q}_2 \boldsymbol{B}. \tag{4.48}$$

Multiplying the last equation from the left with $\boldsymbol{Q}_1^T$ yields

$$\boldsymbol{A} = \boldsymbol{Q}_1^T \boldsymbol{X}_2, \tag{4.49}$$

as $\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$ are required to be orthogonal with respect to each other. The matrices $\boldsymbol{Q}_2$ and $\boldsymbol{B}$ can then be found by a QR decomposition of $\boldsymbol{P} = \boldsymbol{X}_2 - \boldsymbol{Q}_1(\boldsymbol{Q}_1^T \boldsymbol{X}_2)$, i.e.

$$\boldsymbol{Q}_2 \boldsymbol{B} = \boldsymbol{P} = \boldsymbol{X}_2 - \boldsymbol{Q}_1(\boldsymbol{Q}_1^T \boldsymbol{X}_2) \tag{4.50}$$

or any other procedure that yields $\boldsymbol{Q}_2$ as an orthogonal basis of the range of $\boldsymbol{P}$, and then projecting with $\boldsymbol{B} = \boldsymbol{Q}_2^T \boldsymbol{P}$. This leads to the algorithm for the QR update as given in Alg. 4.3.

**Algorithm 4.3** Simple orthogonal decomposition update

**Input:** matrix $\boldsymbol{X} = \begin{bmatrix} \boldsymbol{X}_1 & \boldsymbol{X}_2 \end{bmatrix}$, where $\boldsymbol{X}_1$ has orthogonal columns

**Output:** matrices $\boldsymbol{Q}$, $\boldsymbol{R}$

1: Set $\boldsymbol{Q}_1 \leftarrow \boldsymbol{X}_1$
2: Set $\boldsymbol{A} \leftarrow \boldsymbol{Q}_1^T \boldsymbol{X}_2$
3: Set $\boldsymbol{P} \leftarrow \boldsymbol{X}_2 - \boldsymbol{Q}_1 \boldsymbol{A}$
4: Set $\boldsymbol{Q_2} \leftarrow \textsc{Orth}(\boldsymbol{P})$
5: Set $\boldsymbol{B} \leftarrow \boldsymbol{Q}_2^T (\boldsymbol{X}_2 - \boldsymbol{Q}_1 \boldsymbol{A})$
6: Set $\boldsymbol{Q} \leftarrow \begin{bmatrix} \boldsymbol{Q}_1 & \boldsymbol{Q}_2 \end{bmatrix}$
7: Set $\boldsymbol{R} \leftarrow \begin{bmatrix} \boldsymbol{I} & \boldsymbol{A} \\ \boldsymbol{0} & \boldsymbol{B} \end{bmatrix}$

This algorithm in the given form has unfortunately some deficiencies. First, column vectors in $\boldsymbol{X}_2$ can be (almost) pure linear combinations of column vectors of $\boldsymbol{Q}_1 = \boldsymbol{X}_1$, leading to zero or near-zero column vectors in $\boldsymbol{P}$. Second, orthogonalisation of $\boldsymbol{P} = \boldsymbol{X}_2 - \boldsymbol{Q}_1 \boldsymbol{A}$ may reintroduce components that are in the span of $\boldsymbol{Q}_1$, making $\boldsymbol{Q}_2$ and $\boldsymbol{Q}_1$ not "orthogonal enough" with respect to the required precision.

In practice this may lead to spurious dimensions and subsequently to a blow-up of the numerical rank in iterative methods using this algorithm. There are two remedies for mitigating these effects. First, near-zero column vectors in $\boldsymbol{P} = \boldsymbol{X}_2 - \boldsymbol{Q}_1 \boldsymbol{A}$ need to be detected and removed. This is done by selecting only those columns whose norm is not reduced by a factor smaller than some chosen threshold ($10^{-14}$ in the current implementation worked well). Second, by removing components in span($\boldsymbol{Q}_1$) twice, as was also shown to be generally sufficient in [30], $\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$ are kept orthogonal. Including both steps into the algorithm then leads to Alg. 4.4. An implementation of this algorithm can be

found in sglib in `linalg/qr_internal`.

**Algorithm 4.4** Orthogonal decomposition update

**Input:** matrix $\boldsymbol{X} = \begin{bmatrix}\boldsymbol{X}_1 & \boldsymbol{X}_2\end{bmatrix}$, where $\boldsymbol{X}_1$ has orthogonal columns

**Output:** matrices $\boldsymbol{Q}$, $\boldsymbol{R}$

1: Set $\boldsymbol{Q}_1 \leftarrow \boldsymbol{X}_1$
2: Set $\boldsymbol{A} \leftarrow \boldsymbol{Q}_1^T \boldsymbol{X}_2$
3: Set $\boldsymbol{P} \leftarrow \boldsymbol{X}_2 - \boldsymbol{Q}_1 \boldsymbol{A}$
4: Select only those columns of $\boldsymbol{P}$ into $\tilde{\boldsymbol{P}}$, such that the ratio of the norm of the column vector in $\tilde{\boldsymbol{P}}$ to that of the corresponding column vector in $\boldsymbol{X}_2$ is greater than some threshold (e.g. $10^{-14}$).
5: Set $\boldsymbol{Q_2} \leftarrow \text{ORTH}(\tilde{\boldsymbol{P}})$
6: Set $\boldsymbol{Q}_2 \leftarrow \boldsymbol{Q}_2 - \boldsymbol{Q}_1(\boldsymbol{Q}_1^T \boldsymbol{Q}_2)$
7: Set $\boldsymbol{Q_2} \leftarrow \text{ORTH}(\boldsymbol{Q}_2)$
8: Set $\boldsymbol{B} \leftarrow \boldsymbol{Q}_2^T (\boldsymbol{X}_2 - \boldsymbol{Q}_1 \boldsymbol{A})$
9: Set $\boldsymbol{Q} \leftarrow \begin{bmatrix}\boldsymbol{Q}_1 & \boldsymbol{Q}_2\end{bmatrix}$
10: Set $\boldsymbol{R} \leftarrow \begin{bmatrix}\boldsymbol{I} & \boldsymbol{A} \\ \boldsymbol{0} & \boldsymbol{B}\end{bmatrix}$

**Remark 4.4.** *In practice it is necessary to perform the second orthogonalisation with higher precision than the first one, since otherwise still spurious dimensions will be encountered. For the first orthogonalisation this is not necessary. In the Matlab® implementation the first orthogonalisation is thus performed by* `qr`*, while the second one is performed by* `orth`*, which is slower but more accurate than* `qr`*, as it is based internally on the SVD.*

**Remark 4.5.** *Routines for updating QR factorisations with implementations in Fortran 77, including modification as well as insertion and deletion of columns, are summarised in the excellent report by Hammarling and Lucas [40]. For the special case of rank-1 updates to the SVD where the number of updates is large see also the paper by Brand [14], in which a decomposition into*

*five matrices is proposed, such that the outer orthogonal matrices need not be rotated on each update.*

### Non-continuity of the truncation operator

In the previous sections the truncation operators $T_{R'}$ and $T_\varepsilon$ were introduced. For the investigation of iterative methods that are perturbed by those truncations it is an important question, whether those operators are continuous. However, it is easy to see that neither $T_{R'}$ nor $T_\varepsilon$ in general are continuous. As an example, take the family of matrices

$$\boldsymbol{A}_\delta = \begin{bmatrix} 1 & 0 \\ 0 & 1+\delta \end{bmatrix}$$

and consider fixed rank truncation. For target rank $R' = 1$ the truncation operator $T_{R'}$ will have a discontinuity at $\delta = 0$. Namely, for $\delta < 0$ we get $T_{R'}\boldsymbol{A}_\delta = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$ and for $\delta > 0$ we get $T_{R'}\boldsymbol{A}_\delta = \begin{bmatrix} 0 & 0 \\ 0 & 1+\delta \end{bmatrix}$. If we consider truncation to some fixed $\varepsilon$, then $T_\varepsilon$ will have discontinuities at $\delta = \varepsilon$ and at $\delta = -\varepsilon$. For $|\delta| \leq \varepsilon$ the truncation rank will be $R' = 2$, i.e. no truncation is performed at all. Outside this range, i.e. $|\delta| > \varepsilon$, the truncation rank will be $R' = 1$ with the same results as in the example of fixed rank truncation.

## 4.2 Higher order tensors

Representations of tensors of order higher than two exhibit some difficulties that are not present for second order tensors. As those difficulties are essentially the same as long as the order is *higher* than two, the presentation will be limited to third order tensors, as this considerably simplifies notation. A good overview on decompositions for higher order tensors can be found in the review article by Kolda and Bader [51].

For higher order tensors one would like to keep the nice properties that are guaranteed by the SVD for second order. I.e. given a tensor $\mathbf{x}$ one would like to have a representation

$$\mathbf{x} := \sum_{i=1}^{R} \sigma_i \boldsymbol{x}_i \otimes \boldsymbol{y}_i \otimes \boldsymbol{z}_i \tag{4.51}$$

such that the spanning vectors are pairwise orthogonal or at least independent. If $\mathbf{x}$ is represented by

$$\mathbf{x} := \sum_{i,j,k=1}^{R_1,R_2,R_3} s_{ijk}\, \boldsymbol{x}_i \otimes \boldsymbol{y}_j \otimes \boldsymbol{z}_k \tag{4.52}$$

the spanning vectors can indeed be chosen to be orthogonal. This format is called the Tucker format; $s_{ikj}$ is itself a tensor of order three and is called the *core tensor*. A generalisation of the SVD to third order tensors would thus mean diagonalising the core tensor $s_{ijk}$ *and* having the spanning vectors pairwise orthogonal. This, however, is not possible in general [51]. Due to this difficulty various formats have been developed and are currently a field of active research. Some of the formats thus developed shall be briefly discussed.

For the canonical format the tensor approximation problem is not well posed as the subset of rank $R$ tensors is generally not closed for $R > 1$ [51]. Furthermore, even if a best rank $R$ approximation exists, the best rank $R+1$ approximation can in general not be computed from there. Heuristic approaches to compute a rank $R$ approximation, which is not necessarily the best, are mostly based on the alternating least squares methods (ALS). However, the ALS method is very slow compared to the truncation algorithm for second order tensors, and furthermore not guaranteed to converge at all [51].

The canonical format using ALS for truncation from the Tensor Toolbox [6, 7] has been tried by the author on stochastic systems. While it did work for small systems, for larger systems the runtime needed for the truncations made the method very inefficient. More efficient approaches based on solving a minimisation problem

have been developed by Espig [22], but have not been studied on stochastic systems yet.

In contrast to the canonical format a best approximation does exist in the Tucker format. Truncations in the Tucker format are generally performed by ALS and high order SVD (HOSVD) [55]. One problem with the Tucker format in higher dimensions is that the size of the core tensor itself grows exponentially. Thus, other representations have been developed recently, e.g. so-called tensor trains [72, 73], or the $\mathcal{H}$-Tucker format [33], but are beyond the scope of this thesis.

## 4.3 Motivation for tensor product solvers

As could be seen in the previous sections, tensor approximations can significantly reduce memory and runtime costs. The results have been summarised in Table 4.1. Clearly, as long as the representation rank $R$ is small in comparison to $N$ and $M$, the savings in storage and runtime can be compelling.

| Format | full/matrix | canonical |
|---|---|---|
| Memory | $\Theta(MN)$ | $\Theta(R(M+N))$ |
| Operator application | $\tau_A\Theta(N) + \tau_B\Theta(M)$ | $(\tau_A + \tau_B)\Theta(R)$ |
| Vector space ops. $(+, \cdot)$ | $\Theta(MN)$ | $\mathrm{O}(R(M+N))$ |
| Truncations QR | | $\Theta(R^2(M+N))$ |
| Truncations SVD | | $\Theta(R^3)$ |

**Table 4.1:** Comparison of memory and runtime costs between the full (matrix) and canonical representations for tensors in $\mathbb{R}^M \otimes \mathbb{R}^N$. $R$ denotes the representation rank of the tensor in canonical format. $\tau_A$ and $\tau_B$ denote the typical operation counts for matrix multiplication with matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ for a given tensor product operator $\boldsymbol{A} \otimes \boldsymbol{B}$.

To illustrate the relative magnitudes of the quantities in Table 4.1 some rough back-of-the-envelope calculations shall be given here. As only orders of magnitude are important here for comparison all values are given simply as powers of 10.

In the models used in this thesis values for $N$ and $M$ are about $10^4$ (see Section 8.1). So storing a full tensor takes about $10^8$ units of memory, typical vector space operations take about $10^8$ operations. Operator applications take roughly $10^4\tau \approx 10^8$ operations, where the time per operator application $\tau$ has been approximated by $10^4$, since the operation count for application of the matrices involved is roughly the size of the matrix times a small number (see e.g. [89, p. 10]).

- If the numerical rank is small, say $R = 10$, then in the canonical format, storage takes about $10^5$ units of memory, and vector space operations and operator applications take about $10^5$ operations. Additionally, there are QR and SVD factorisations, which take about $10^6$ and $10^3$ operations, respectively. Overall, storage and runtime are reduced by about three orders of magnitude.

- If the numerical rank is larger, say $R = 100$, storage takes about $10^6$ units of memory, and vector space operations and operator applications take about $10^6$ operations. The QR and SVD factorisations, however, take now about $10^8$ and $10^6$ operations, respectively. Storage is still reduced by a large factor, however, runtime is approximately in the range as for the full format, due to the time spent in the QR decompositions.

- If the numerical rank is large, say $R = 1000$, storage takes about $10^7$ units of memory and thus still less than the full format. The QR and SVD factorisations, however, take now about $10^{10}$ and $10^9$ operations, making the method computationally unattractive.

Though the estimates above are very rough, as any constants have been omitted (and would have depended on details of the problem, anyway), the scaling behaviour is clearly visible.

It is further consistent with the findings from the numerical experiments (see Section 8), that for systems with low rank solutions tensor methods can outperform methods based on the full tensor

format. However, if the rank is high and the problem can still be computed in a full tensor format the full format solver is usually faster. The main area of application for those low-rank methods is thus, when either the rank is very low, or if the dimensions of the problem are such that computation in the full format is not feasible.

In applications involving elliptic stochastic PDEs the random fields of the solution often show low rank due to the smoothing properties of the operator. Thus it would be advantageous to use a low-rank format throughout the process of solving the SPDE, i.e. if the input fields are given in a low-rank format as can be achieved by the KLE for example, this format should be kept through the whole solution process, to finally yield a low-rank solution to the SPDE itself. In this manner, problems can be computed that would not be feasible to compute in the full format due to memory constraints. If the solution has furthermore low rank the methods developed in this thesis can be expected to speed up the solution process significantly.

# Chapter 5

# Perturbed iterative processes

*This chapter discusses convergence issues of iterative solvers based on tensor product methods. In the first section basic facts about iterative methods in general are recollected and how they are affected by the tensor approximations. Most of this can be done without reference to specific solvers or tensor formats, and be regarded abstractly as perturbations of general iterative processes. As the perturbed iterations tend not to converge, but to stagnate in some neighbourhood of the solution, a procedure to detect this stagnation is developed. The chapter concludes with a scheme to dynamically select the truncation parameter and hence the perturbation in order improve the performance of tensor approximation based methods.*

## 5.1 Iterative processes

In this section a few basic results about iterative processes shall be recollected. For more elaborate treatments see e.g. [9, 36, 89]. The discretisation of linear (stochastics) PDEs usually results in large systems of linear equations of the form

$$Ax = b \tag{5.1}$$

where $x$ and $b$ are elements of some finite dimensional normed vector space $\mathcal{V}$ and $A : \mathcal{V} \to \mathcal{V}$ is a non-singular linear operator. In general, iterative solvers turn Eq. (5.1) into an iterative process of the form

$$x^{(k+1)} = \Phi^{(k)}(x^{(k)}) \tag{5.2}$$

such that the solution $x^* = A^{-1}b$ of Eq. (5.1) is the unique stable fixed point of Eq. (5.2). In other words, $x^*$ is the only solution to $x = \Phi^{(k)}(x)$ for every fixed $k$. In case the iteration map $\Phi^{(k)}$ depends on $k$ the process is called *instationary*, otherwise it is called *stationary* and the index can be dropped. The iterates of the process will form a sequence which will be denoted by $(x^{(k)})_{k\geq 0} = (x^{(0)}, x^{(1)}, x^{(2)}, \ldots)$. The error at iteration $k$ will be denoted by $e^{(k)}$, i.e.

$$e^{(k)} = x^* - x^{(k)}. \tag{5.3}$$

Some results about convergence of iterative processes of the form Eq. (5.2) and corresponding error estimates are summarised in the following (see e.g. [9, 19, 71, 89]).

### Convergence of stationary methods

For stationary methods with $\Phi^{(k)} = \Phi$ convergence of the sequence $(x^{(k)})_{k\geq 0}$ can be established by using the contraction mapping principle (also known as the Banach fixed point theorem). If $\Phi$ is a *contraction*, i.e. if there is a constant $q < 1$, such that for all $x, y \in \mathcal{V}$ it holds that

$$\|\Phi(x) - \Phi(y)\| \leq q\|x - y\|, \tag{5.4}$$

the contraction mapping principle asserts that $\Phi$ has a unique fixed point $x^*$, and the sequence of iterates $x^{(k)}$ converges at least linearly to $x^*$ (see e.g. [71, Theorem 5.1.3]). For the $k$-th iterate the a priori error estimate

$$\|e^{(k)}\| \leq q^k\|e^{(0)}\|, \tag{5.5}$$

can be established [71]. Though the a priori error estimate gives bounds on the rate of convergence, it not practical for computations, since there are quantities involved that are not computable, namely the initial error $e^{(0)}$. Of greater practical value is the a

posteriori error estimate

$$\|e^{(k)}\| \leq \frac{q}{1-q}\|x^{(k)} - x^{(k-1)}\|. \tag{5.6}$$

For a derivation see e.g. [71, Theorem 12.1.2]. If the constant of contractivity $q$ is known, an estimate or upper bound for the error can thus be computed. If $q$ is not known exactly, it can be estimated before or during the iterative process itself.

### Convergence of instationary methods

For instationary methods the case is slightly more difficult. Since the $\Phi^{(k)}$ may be all different, there is no theorem that guarantees the existence of a unique fixed point. In the methods discussed, however, the construction of the $\Phi^{(k)}$ is such that each has the same fixed point, namely the solution of Eq. (5.1) (see Remark 5.1).

**Remark 5.1.** *For many iterative processes, including the ones discussed in this thesis, the iteration maps $\Phi^{(k)}$ can be written in the form*

$$\Phi^{(k)}(x) = x + \Psi^{(k)}(r) \tag{5.7}$$

*with $r = b - Ax$ and $\Psi^{(k)}(r) = 0$ if and only if $r = 0$. It follows immediately that $x$ is a fixed point of $\Phi^{(k)}$ if and only if $r = 0$, i.e. if $x$ is a solution to $Ax = b$.*

For instationary methods we can prove the following theorem.

**Theorem 5.2.** *Let $\left(\Phi^{(k)}\right)_{k\geq 0}$ be a family of* uniformly contractive *maps, i.e. there is a $q < 1$ such that for all $x, x^* \in \mathcal{V}$ and all $k > 0$ it holds that*

$$\|\Phi^{(k)}(x) - x^*\| \leq q\|x - x^*\|, \tag{5.8}$$

*then the sequence $(x^{(k)})_{k\geq 0}$ will also be at least linearly convergent, and the same error estimates as in Eq. (5.5) and in Eq. (5.6) hold.*

*Proof.* See e.g. Theorem 11.1.2 in [71]. □

Most instationary methods, like e.g. the Krylov subspace methods, attain superlinear order of convergence, which is essentially the reason for the construction those methods[1].

## 5.2 Perturbed iterations

Section 4.1.4 described how the numerical rank of a tensor in the canonical format grows with each addition, and that the tensor therefore has to be truncated frequently to keep memory and runtime moderate. This truncation—when happening inside an iterative method—can be regarded as a perturbation to the sequence of iterates given by Eq. (5.2).

An important question is, how these perturbations affect convergence and error estimates in relation to the unperturbed sequence. This change can be studied in an abstract way without resorting to the details of specific linear solvers and of the tensor format. Rather, the effect of perturbations on a general iterative process will be analysed.

Further, as it can be expected that the perturbed iterations will not converge, but in the best case stagnate in some neighbourhood of the solution, an indicator that signals the occurrence of this stagnation needs to be developed.

Perturbed iterations have been discussed at some places in the literature. Many basic results for *approximate contractions* can be found in the book by Ortega and Rheinboldt [71, Section 12.2]. Hackbusch et al. [38] have addressed this problem for approximate iterations on matrices and superlinearly convergent methods. The results there are quite strong, however, the preconditions for those results to hold are generally difficult to satisfy. Furthermore, the methods analysed here are to the most part only linearly

[1] Theoretically, since Krylov subspace methods are direct methods in finite dimensions, the concept of convergence does not really apply. In practice, however, those methods are used as iterative methods and superlinear convergence can be observed experimentally [83].

convergent, so that those results do not apply directly. Otherwise, the topic, especially in the context of the effects of rounding errors or floating point precision issues has mostly been discussed in respect to specific algorithms. Especially in the area of Krylov subspace methods there is a lot of literature to be found on the subject, see e.g. [35]. Most of the following results, except for the dynamic truncations, have already been published by Matthies and the author in [63, 64].

### 5.2.1 Error estimates

#### Fixed perturbations

When the iterates are represented in a tensor format like the canonical representation the iterates need to be truncated after each step, resulting in a perturbation of the original sequence $x^{(k)}$ to a new sequence $\tilde{x}^{(k)}$, i.e.

$$\tilde{x}^{(k+1)} = \left(T_\varepsilon \circ \Phi^{(k)}\right)(\tilde{x}^{(k)}), \tag{5.9}$$

and $\tilde{x}^{(0)} = T_\varepsilon x^{(0)}$. The perturbation effected by the truncation algorithm can be described by a perturbation operator $T_\varepsilon$, characterised by the property that

$$\|T_\varepsilon x - x\| \leq \varepsilon \tag{5.10}$$

for all $x \in \mathcal{V}$. Since in this case the maximal magnitude of the perturbation does not depend on $x$, this kind of perturbation is called *fixed perturbations*. Relative perturbations, in which the magnitude of the perturbation depends on the magnitude of $x$, are treated in the next section.

Often the truncation is performed already during or inside the iteration operator resulting in a perturbed operator $\Phi_\varepsilon$. If the operator is still contractive with contractivity $q$, the results derived in the following are still valid if the perturbed operator fulfils

$$\|\Phi_\varepsilon^{(k)}(x) - \Phi^{(k)}(x)\| \leq \varepsilon. \tag{5.11}$$

The case in which the perturbation is applied only after the iteration operator can be subsumed under this case by setting $\Phi_\varepsilon^{(k)} = T_\varepsilon \circ \Phi^{(k)}$.

The questions that arise now are, what the limiting behaviour of the perturbed sequence will be. For example, will the sequence still converge, or will it stagnate in some vicinity of the fixed point? What can be said about a priori and a posteriori error estimates? To answer the question about convergence, first a theorem on the a priori error will be given (see [63]).

**Theorem 5.3.** *Let $\mathcal{V}$ be a Banach space and $\Phi^{(k)} : \mathcal{V} \to \mathcal{V}$ a family of uniformly contractive mappings, i.e. for all $k$ there is some $q < 1$ such that $\|\Phi^{(k)}(x) - \Phi^{(k)}(y)\| \leq q\|x - y\|$ and all $\Phi^{(k)}$ have the same fixed point $x^*$. Let $\left(x^{(k)}\right)_{k\geq 0} \in \mathcal{V}$ be a sequence generated by some $x^{(0)} \in \mathcal{V}$ and $x^{(k+1)} = \Phi^{(k)}\left(x^{(k)}\right)$. Let $\Phi_\varepsilon^{(k)}$ be a perturbed operator such that $\|\Phi_\varepsilon^{(k)}(x) - \Phi^{(k)}(x)\| \leq \varepsilon$, and let $\left(\tilde{x}^{(k)}\right)_{k\geq 0} \in \mathcal{V}$ be a perturbed sequence generated by $\tilde{x}^{(0)} = x^{(0)}$ and $\tilde{x}^{(k+1)} = \Phi_\varepsilon^{(k)}(\tilde{x}^{(k)})$. Then the a priori error estimate*

$$\|\tilde{x}^{(k)} - x^*\| \leq q^k \|x^{(0)} - x^*\| + \varepsilon \frac{1 - q^k}{1 - q}. \tag{5.12}$$

*holds for all $k \geq 0$.*

*Proof.* In the following let the difference between the original and the perturbed sequence be denoted by

$$\delta^{(k)} = x^{(k)} - \tilde{x}^{(k)} \tag{5.13}$$

and the error made in the perturbed sequence by

$$\tilde{e}^{(k)} = x^* - \tilde{x}^{(k)}. \tag{5.14}$$

The error $\tilde{e}^{(k)}$ in the perturbed sequence can be written as the sum of the perturbation error $\delta^{(k)}$ and the error of the unperturbed

sequence $e^{(k)}$. Using the triangle inequality

$$\|\tilde{e}^{(k)}\| \leq \|e^{(k)}\| + \|\delta^{(k)}\| \tag{5.15}$$

the components of the error can be estimated separately. The first term has already been estimated by $\|e^{(k)}\| \leq q^k \|e^{(0)}\|$ (see Eq. (5.5)). For the second term we see that

$$\begin{aligned}
\|\delta^{(k+1)}\| &= \|\tilde{x}^{(k+1)} - x^{(k+1)}\| \\
&= \|\Phi_\varepsilon^{(k)}(\tilde{x}^{(k)}) - \Phi^{(k)}(x^{(k)})\| \\
&\leq \|\Phi_\varepsilon^{(k)}(\tilde{x}^{(k)}) - \Phi^{(k)}(\tilde{x}^{(k)})\| + \|\Phi^{(k)}(\tilde{x}^{(k)}) - \Phi^{(k)}(x^{(k)})\| \\
&\leq \varepsilon + q\|\delta^{(k)}\|
\end{aligned} \tag{5.16}$$

using the contractivity of $\Phi^{(k)}$ and the fact that the perturbation of $\Phi_\varepsilon^{(k)}$ is limited by $\varepsilon$ (see Eq. (5.11)). Using this formula inductively on $\delta^{(0)} = 0$ it follows that

$$\|\delta^{(k)}\| \leq \varepsilon(1 + q + \cdots + q^{k-1}) = \varepsilon \frac{1 - q^k}{1 - q}, \tag{5.17}$$

where in the last equality the well-known formula for the geometric series has been used (see e.g. [16]). Combining both terms proves the theorem. □

Hence the error is composed of a part that tends to zero for large $k$ and another part proportional to the perturbation $\varepsilon$ that will not vanish for large $k$. Letting $k$ go to infinity in the preceding theorem leads immediately to the following corollary.

**Corollary 5.4.** *Let the assumptions be as in Theorem 5.3, then it holds that*

$$\limsup_{k \to \infty} \|\tilde{x}^{(k)} - x^*\| \leq \frac{\varepsilon}{1 - q}. \tag{5.18}$$

So for large $k$ we can expect the iterates to stagnate in a neighbourhood of size $\delta_\varepsilon = \varepsilon/(1 - q)$ around the true solution $x^*$. How this stagnation can be detected will be discussed in Section 5.2.2. First, an estimate for the a posteriori error shall be given.

**Theorem 5.5.** *Let the assumptions be as in Theorem 5.3, then the a posteriori error estimate*

$$\|\tilde{x}^{(k)} - x^*\| \leq \frac{q}{1-q}\|\tilde{x}^{(k)} - \tilde{x}^{(k-1)}\| + \frac{\varepsilon}{1-q}. \tag{5.19}$$

*holds.*

*Proof.* The a posteriori estimate for the unperturbed sequence Eq. (5.6) can be rewritten in the form

$$\|\Phi^{(k-1)}(x) - x^*\| \leq \frac{q}{1-q}\|\Phi^{(k-1)}(x) - x\|.$$

Then it follows that

$$\begin{aligned}
\|\tilde{e}^{(k)}\| &\leq \|\tilde{x}^{(k)} - \Phi^{(k-1)}(\tilde{x}^{(k-1)})\| + \|\Phi^{(k-1)}(\tilde{x}^{(k-1)}) - x^*\| \\
&\leq \|\Phi_\varepsilon^{(k-1)}(\tilde{x}^{(k-1)}) - \Phi^{(k-1)}(\tilde{x}^{(k-1)})\| + \\
&\quad \|\Phi^{(k-1)}(\tilde{x}^{(k-1)}) - x^*\| \\
&\leq \varepsilon + \frac{q}{1-q}\|\Phi^{(k-1)}(\tilde{x}^{(k-1)}) - \tilde{x}^{(k-1)}\| \\
&\leq \varepsilon + \frac{q}{1-q}\Big(\|\Phi^{(k-1)}(\tilde{x}^{(k-1)}) - \Phi_\varepsilon^{(k-1)}(\tilde{x}^{(k-1)})\| + \\
&\qquad\qquad \|\Phi_\varepsilon^{(k-1)}(\tilde{x}^{(k-1)}) - \tilde{x}^{(k-1)}\|\Big) \\
&\leq \frac{\varepsilon}{1-q} + \frac{q}{1-q}\|\tilde{x}^{(k)} - \tilde{x}^{(k-1)}\|,
\end{aligned}$$

proving the theorem. □

If the contractivity constant $q$ is known or can be estimated this estimate can be used in practical computations, serving for example as a termination criterion.

### Relative perturbations

In practice it is often preferable not to truncate to some fixed $\varepsilon$, but rather relative to the size of the current iterate. In this case the preceding theorems have to be modified. If the perturbation

effected can be described by the perturbed family of operators $(\Phi_\varepsilon^{(k)})_{k\geq 0}$ for which the property

$$\|\Phi_\varepsilon^{(k)}(x) - \Phi^{(k)}(x)\| \leq \varepsilon\|x\| \tag{5.20}$$

holds, the following estimates can be derived.

**Theorem 5.6.** *Let the conditions hold as in Theorem 5.3 except that for the perturbation operator it holds that* $\|\Phi_\varepsilon^{(k)}(x) - \Phi^{(k)}(x)\| \leq \varepsilon\|x\|$. *Further let* $c_x = \sup_{k\geq 0}\|x^{(k)}\|$ *and* $\tilde{q} = q+\varepsilon < 1$. *Then*

$$\|\tilde{x}^{(k)} - x^*\| \leq \tilde{q}^k \|x^{(0)} - x^*\| + \varepsilon c_x \frac{1-\tilde{q}^k}{1-\tilde{q}} \tag{5.21}$$

*holds for all* $k \geq 0$.

*Proof.* The proof is essentially the same as in the case of fixed perturbations. Only in the estimate for $\|\delta^{(k+1)}\|$ in Eq. (5.16) explicit reference to the perturbation operator was made. The estimate for $\|\delta^{(k+1)}\|$ for relative perturbations then changes here to

$$\begin{aligned}\|\delta^{(k+1)}\| &\leq \varepsilon\|\tilde{x}^{(k)}\| + q\|\delta^{(k)}\| \\ &= \varepsilon\|x^{(k)} + \delta^{(k)}\| + q\|\delta^{(k)}\| \\ &\leq \varepsilon\|x^{(k)}\| + (q+\varepsilon)\|\delta^{(k)}\|\end{aligned} \tag{5.22}$$

As the sequence $x^{(k)}$ is convergent, it can be bounded by some $c_x$ such that $\|x^{(k)}\| \leq c_x$ for all $k > 0$. The estimate for $\|\delta^{(k+1)}\|$ is thus the same as for fixed perturbation only with $c_x\varepsilon$ replaced for $\varepsilon$ and $\tilde{q} = q + \varepsilon$ replaced for $q$. □

As for the fixed perturbations the following corollary follows immediately.

**Corollary 5.7.** *Let the assumptions be as in Theorem 5.6, then it holds that*

$$\limsup_{k\to\infty}\|\tilde{x}^{(k)} - x^*\| \leq \frac{\varepsilon c_x}{1-\tilde{q}}. \tag{5.23}$$

So, in order to guarantee stagnation for relative truncation, we need to limit $\varepsilon$ to $1 - q$ or combine it with fixed truncation. However, the update ratio and dynamic truncation that will be introduced in the next two sections, also provide a measure to protect against cases like this.

Though the proof is trivial, the a posteriori error estimate for relative perturbations shall also be given as a theorem here.

**Theorem 5.8.** *Let the assumptions be as in Theorem 5.6. Then the a posteriori error estimate*

$$\|\tilde{x}^{(k)} - x^*\| \leq \frac{q}{1-q}\|\tilde{x}^{(k)} - \tilde{x}^{(k-1)}\| + \frac{\varepsilon\|\tilde{x}^{(k-1)}\|}{1-q}. \tag{5.24}$$

*holds.*

*Proof.* As the a posteriori estimate depends only on the one step taken from $\tilde{x}^{(k-1)}$ to $\tilde{x}^{(k)}$ the estimate is the same as for a fixed perturbation of size $\varepsilon\|\tilde{x}^{(k-1)}\|$. Inserting $\varepsilon\|\tilde{x}^{(k-1)}\|$ for $\varepsilon$ in Eq. (5.19) gives Eq. (5.24). □

This error estimate has been tested in practice and, especially in the case of simple iterative solvers, proved to give good upper bounds for the error (see Section 8.3 on page 124).

**Remark 5.9.** *Note that in the methods actually implemented only relative truncations (perturbations) were performed. Usually it is not sensible to perform truncation to some fixed $\varepsilon$, irrespective of the norm of the iterate $x$, which would make sense only when the norm of the fixed point were known a priori. However, when the iterates approach the fixed point, the truncation with fixed and with relative $\varepsilon$ become more or less equivalent, with $\varepsilon_{\text{abs}} = \|x^*\|\varepsilon_{\text{rel}}$.*

### 5.2.2 Detection of stagnation

It is common for iterative processes that the steps $\tilde{x}^{(k+1)} - \tilde{x}^{(k)}$ are relatively large during the first iterations and tend to get consecutively smaller as the exact solution is approached. Therefore, the influence of the perturbations will be very small in the

beginning and grow with every iteration until it is comparable in size with the actual update, which becomes evident in the a priori estimates Eq. (5.12) and Eq. (5.21). In this case it is futile to further continue the iteration, since every progress is undone by the truncation. There are two alternatives that can be chosen from: either stop the iteration at this point and return the current iterate as the best possible given the current truncation parameter, or reduce the truncation parameter by a certain factor and continue the iteration until the iteration goal has been achieved (see Section 5.2.3).

Either way, it has to be determined whether this stagnation regime has been reached. Stagnation caused by truncation can be detected by comparing the steps the iterative process would take with and without truncation. As first case we will consider perturbed operators that can be decomposed as $\Phi_\varepsilon^{(k)} = T_\varepsilon \circ \Phi^{(k)}$. Then the steps of the truncated iterations can be dissected into the following sequence:

1. Compute the next iterate without truncation:
$$x^{(k+1)} = \Phi^{(k)}(\tilde{x}^{(k)}) \tag{5.25}$$
2. Compute the proposed step that would be taken in the absence of truncation:
$$\Delta x^{(k)} = x^{(k+1)} - \tilde{x}^{(k)} \tag{5.26}$$
3. Perform the truncation:
$$\tilde{x}^{(k+1)} = T_\varepsilon(x^{(k+1)}) \tag{5.27}$$
4. Compute the actual step that is taken by the perturbed iterative process:
$$\Delta \tilde{x}^{(k)} = \tilde{x}^{(k+1)} - \tilde{x}^{(k)} \tag{5.28}$$

In order to detect stagnation the proposed step $\Delta x^{(k)}$ needs to be compared to the truncated step $\Delta \tilde{x}^{(k)}$. Since not only the size of the step is important, but also its direction it makes sense to take the inner product of the proposed and the truncated step and

further to scale it by the size of the proposed step. This quantity

$$\upsilon^{(k)} = \frac{\langle \Delta\tilde{x}^{(k)} | \Delta x^{(k)} \rangle}{\langle \Delta x^{(k)} | \Delta x^{(k)} \rangle} \tag{5.29}$$

will be called the *update ratio*, since it reflects how much of the proposed update is effectively used. The update ratio should always be around 1 for the iterative process to be efficient. If it is smaller than one not all of the proposed step is taken, while if it is larger the iteration "overshoots" in some way, which is not desirable either, and usually leads to oscillations during the iteration. Thus, as a criterion, of whether the update is still satisfactory, the condition

$$|1 - \upsilon^{(k)}| \leq \delta_{\max} \tag{5.30}$$

can be used, where $\delta_{\max}$ is some threshold for the deviation of the update ratio. Experimentally, setting $\delta_{\max}$ to values between 0.01 and 0.1 produced satisfying results, in that on the one hand iterations are not stopped while still progress is being made, while on the other not many unnecessary iterations are performed, which do not improve the solution.

### Numerical experiments

To see whether the update ratio performs well in practice as an indicator for the deterioration of the convergence of perturbed iterative processes, numerical experiments have been conducted. Fig. 5.1 shows the update ratio and relative error for the model problem (see Section 8.1) and a perturbed iterative process using simple iterations (see Section 6.1.1). It can be seen in the left figure for $\delta_{\max} = 0.1$, that the deviation of the update ratio from 1 passes the threshold (indicated by the dashed black line) approximately where the convergence of the relative error starts to deteriorate. The more conservative choice of $\delta_{\max} = 0.1$ stops a bit earlier, but can sometimes lead to faster convergence, especially in conjunction with dynamic truncation.

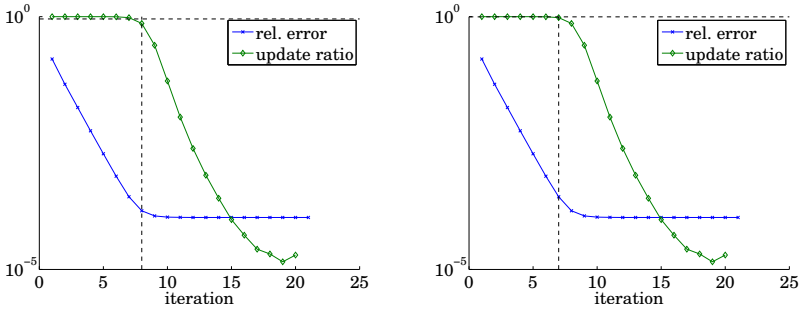



**Figure 5.1:** Relative error and update ratio for the generalised simple iterations. The threshold for the update ratio is $\delta_{\max} = 0.1$ (left) and $\delta_{\max} = 0.01$ (right); the cutoff indicated by the dashed lines. (Script: `figures/show_update_ratio_and_posterior_err`)

**Remark 5.10.** *As noted earlier, the indicator works exactly as proposed only, if the perturbed operator is of the form $T_\varepsilon \circ \Phi^{(k)}$. However, if the operator can be decomposed as $T_{\varepsilon_2} \circ \Phi^{(k)}_{\varepsilon_1}$ the method still works, if $\varepsilon_1 \ll \varepsilon_2$ holds for the respective perturbations, i.e. the perturbations inside the operator are at least an order of magnitude smaller than the perturbations after the update step. This needs to be respected when choosing the truncation parameters in Section 6.1.1.*

### 5.2.3 Dynamic truncation

In the first steps of an iterative process the iterate is generally far away from the solution. Then especially the modes with small amplitude do not contain relevant information about the exact solution and may just as well be discarded. As during the iterative process the iterates come closer to the exact solution also the smaller modes will become important for an accurate representation of the solution and should be retained.

From this rather qualitative observation it can be assumed that during the first stages of the iterative process the truncation may be done with a higher value of $\varepsilon$ as no, or at least not much, relevant information is lost. When the process comes closer to the

exact solution, also the weaker modes contribute to the accuracy of the approximate solution, and should not be truncated, hence requiring a smaller $\varepsilon$.

A reliable and efficiently computable criterion, of whether the information contained in the update step is nearly completely used or whether most of the new information is discarded by the truncation, is the update ratio described in Section 5.2.2. If the update ratio is near one, almost all information contained in the update step has been incorporated in the new iterate. If in contrast the update ratio is significantly different from one, this indicates that a large part of the information contained in the update step has been discarded by the truncation. Thus, in the latter case the truncation parameter $\varepsilon$ should be reduced. As this cannot be done indefinitely for performance reasons and memory constraints, it is usually be performed only until some prescribed minimal truncation parameter $\varepsilon_{\min}$ or maximal numerical rank $k_{\max}$ is reached. The resulting algorithm, combining the ideas of the update ratio and the dynamic truncation, is sketched in Alg. 5.1.

### Error estimates

The convergence results and error estimates of Section 5.2.1 were derived for a constant truncation parameter. In the following these results are adapted for dynamic truncation. As the propositions and proofs are almost identical for fixed and relative perturbations both cases will be handled together.

Suppose $\varepsilon^{(k)}$ is a non-increasing sequence of truncation parameters, chosen e.g. by Algorithm 5.1. Then the following theorem can be proven.

**Theorem 5.11.** *Let the assumptions hold as in Theorem 5.3 and Theorem 5.6, where now for the perturbed operators* $\|\Phi^{(k)}_{\varepsilon^{(k)}}(x) - \Phi^{(k)}(x)\| \leq \varepsilon^{(k)}$ *and* $\|\Phi^{(k)}_{\varepsilon^{(k)}}(x) - \Phi^{(k)}(x)\| \leq \varepsilon^{(k)}\|x\|$*, respectively, holds and* $\left(\varepsilon^{(k)}\right)_{k\geq 0}$ *is a non-increasing sequence of non-negative*

**Algorithm 5.1** Dynamic truncation

**Input:** initial guess $x^{(0)}$, truncation parameter $\varepsilon$, initial truncation parameter $\varepsilon_0$, reduction factor $\alpha$, update ratio threshold $\delta$

$\tilde{\varepsilon} \leftarrow \varepsilon_0$
**for** $k \leftarrow 1, 2, \ldots,$ **do**
  compute exact step: $y^{(k)} \leftarrow \Phi^{(k)}(x^{(k-1)})$
  **loop**
    $x^{(k)} \leftarrow T_{\tilde{\varepsilon}}\, y^{(k)}$
    $\upsilon^{(k)} \leftarrow \langle \Delta x^{(k)} | \Delta y^{(k)} \rangle / \langle \Delta y^{(k)} | \Delta y^{(k)} \rangle$
      with $\Delta y^{(k)} = y^{(k)} - x^{(k-1)}$ and $\Delta x^{(k)} = x^{(k)} - x^{(k-1)}$
    **if** $|\upsilon^{(k)} - 1| < \delta$ **then**
      ▷ Ok, update ratio close enough to 1
      **exit loop**
    **else if** $\tilde{\varepsilon} > \varepsilon$ **then**
      reduce truncation parameter: $\tilde{\varepsilon} \leftarrow \max(\varepsilon, \alpha\tilde{\varepsilon})$
    **else**
      ▷ Error, no further reduction of $\tilde{\varepsilon}$ possible
      **exit for**
    **end if**
  **end loop**
**end for**

*numbers. Then the a priori error estimate*

$$\|\tilde{x}^{(k)} - x^*\| \leq q^{k-s} \|\tilde{x}^{(s)} - x^*\| + \varepsilon^{(s)} \frac{1 - q^{k-s}}{1 - q}. \tag{5.31}$$

*holds for fixed perturbations and*

$$\|\tilde{x}^{(k)} - x^*\| \leq \tilde{q}^{k-s} \|\tilde{x}^{(s)} - x^*\| + \varepsilon^{(s)} c_{x,s} \frac{1 - \tilde{q}^{k-s}}{1 - \tilde{q}}. \tag{5.32}$$

*hold for relative perturbations, where $s \geq 0$ is a non-negative integer.*

*Proof.* All perturbed operators $\Phi^{(k)}_{\varepsilon^{(k)}}$ with $k \geq s$ induce perturbations with magnitude smaller or equal to $\varepsilon^{(s)}$ as $\left(\varepsilon^{(k)}\right)_{k\geq 0}$ is a decreasing sequence. Thus the results for fixed and relative perturbations (see Eq. (5.12) and Eq. (5.21)) can be applied a fortiori with $\varepsilon = \varepsilon^{(s)}$ for the subsequence starting at $k = s$. The constant $c_x$, used in the estimate for the relative perturbations, can then be reduced to $c_{x,s} = \sup_{k\geq s} \|x^{(k)}\| \leq c_x$ as the supremum needs to be taken only over the subsequence. □

For the a posteriori error the following theorem can be proven.

**Theorem 5.12.** *Let the assumptions hold as in Theorem 5.11. Then the a posteriori error estimate*

$$\|\tilde{x}^{(k)} - x^*\| \leq \frac{q}{1 - q} \|\tilde{x}^{(k)} - \tilde{x}^{(k-1)}\| + \frac{\varepsilon^{(k-1)}}{1 - q}. \tag{5.33}$$

*holds for fixed perturbations, and the estimate*

$$\|\tilde{x}^{(k)} - x^*\| \leq \frac{q}{1 - q} \|\tilde{x}^{(k)} - \tilde{x}^{(k-1)}\| + \frac{\varepsilon^{(k-1)} \|\tilde{x}^{(k-1)}\|}{1 - q}. \tag{5.34}$$

*holds for relative perturbations.*

*Proof.* As the a posteriori error estimate depends only on the truncation performed in the last iteration, the estimate Eq. (5.19)

can be used with perturbation size $\varepsilon^{(k-1)}$ and $\varepsilon^{(k-1)}\|\tilde{x}^{(k-1)}\|$ respectively. □

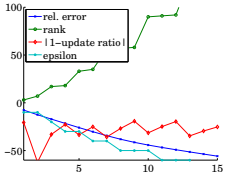
**Remark 5.13.** *Note that in instationary methods like conjugate gradients, though the results still hold as long as every iteration is contractive, the overall convergence may be negatively influenced by the dynamic truncation, as the relatively coarse truncations in the beginning of the iteration may have negative impact on the conjugacy of the sequence.*

### Numerical experiments

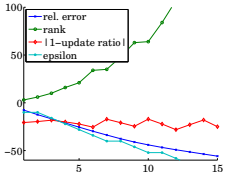
The algorithm has been implemented as part of a simple iterative solver (see `generalised_solve_simple` in sglib for a sample implementation). Experimental results are shown in Fig. 5.2. Here, the dynamic truncation algorithm has been applied as part of the simple iteration solver (gsi) with different parameter choices of $\delta = 0.02$, 0.1 and 0.7 for the update ratio threshold and $\alpha = 1/10$, $1/4$ and $1/2$ for the reduction factor. The initial truncation parameter has been kept constant as $\varepsilon_0 = 0.1$.

As the intention of applying dynamic truncations is to optimise performance of the solver, the best parameter combinations are those, in which the rank grows as slow as possible, while the error drops as fast as possible, preferably as fast as for iterations without truncation. The latter is the case for $\delta = 0.02$ and 0.1, and $\alpha = 1/2$, where the rank growth is slower for $\delta = 0.1$. For $\delta = 0.7$ the rank growth is generally slower as for the other cases, as $\varepsilon$ is decreased less often, however, the decrease in error deteriorates significantly. If the reduction factor is small, like $\alpha = 0.1$, the rank increases too much in one step leading to plateaus in the rank curve, as can be seen in the left column in Fig. 5.2.

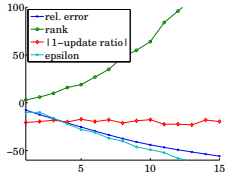
From the cases shown in Fig. 5.2, the best combination of parameters is thus $\delta = 0.1$ and $\alpha = 1/2$, which has been used as the default in all later numerical experiments.



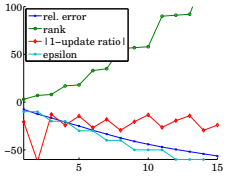

(a)
$\delta = 0.02, \alpha = 1/10$



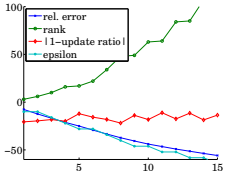

(b)
$\delta = 0.02, \alpha = 1/4$



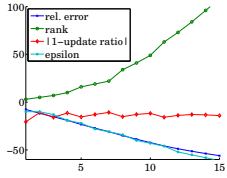

(c) $\delta = 0.02, \alpha = 1/2$



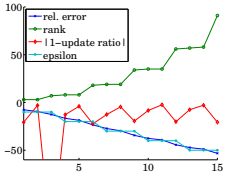

(d)
$\delta = 0.10, \alpha = 1/10$



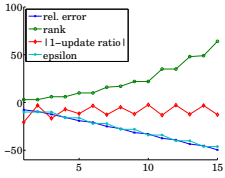

(e) $\delta = 0.10, \alpha = 1/4$



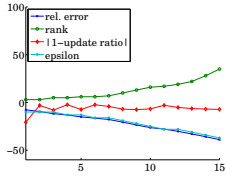

(f) $\delta = 0.10, \alpha = 1/2$




(g)
$\delta = 0.70, \alpha = 1/10$




(h)
$\delta = 0.70, \alpha = 1/4$




(i) $\delta = 0.70, \alpha = 1/2$

**Figure 5.2:** Comparison of the performance of the dynamic truncation algorithm for different parameter choices $\delta = 0.02$, 0.1 and 0.7 and $\alpha = 1/10$, 1/4 and 1/2. Shown are the rank of the current iterate, the relative error, the deviation of the update ratio from one and the current truncation parameter per iteration. Note: The scale on the left is only valid for the rank. The other quantities have logarithmic scaling that is multiplied by a factor such that the qualitative behaviour becomes visible. The scaling factors and the axis limits have been chosen the same for all graphs to facilitate comparison of the effects of parameter choice.

(Script: `figures/show_dynamic_truncation_stats`)

# Chapter 6

# Tensor product solvers

*The effects of perturbations induced by tensor product methods on convergence have been discussed in an abstract setting in the preceding chapter. Further, algorithms to detect stagnation and dynamically choose the truncation parameter have been developed. In this chapter these results are transferred to the implementation of tensor methods in concrete iterative methods like simple iterations and the conjugate gradient method.*

## 6.1 Simple iterations

*Stationary methods*, also known as *simple iterations* or the *preconditioned Richardson method*, are treated exhaustively in the literature, see e.g. [9, 36, 46]. Here some basic facts that will be needed later for the extension to tensor methods will be recollected.

In stationary methods the iteration operator $\Phi^{(k)} = \Phi$ is the same for all iterations. If $\Phi$ is a linear operator it can be represented by a matrix, and it is easy to show that each iteration of this kind has to take the form

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \boldsymbol{P}^{-1}(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(k)}). \tag{6.1}$$

The matrix $\boldsymbol{P}$, called the preconditioner, is usually chosen such that the iterations converge as fast as possible to the solution $\boldsymbol{x}^*$ and the cost of solving with $\boldsymbol{P}$ is relatively cheap. Common choices for $\boldsymbol{P}$ are diagonal or triangular matrices, incomplete factorisations or multigrid iterations (see e.g. [32, 36]). If $\boldsymbol{A}$ is a

tensor operator, different choices of $\boldsymbol{P}$ are common, which will be discussed in Section 7.

In some cases—for analysis and discussion of implementation issues—it is better to separate Eq. (6.1) into the two equations

$$\boldsymbol{r}^{(k)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(k)} \tag{6.2}$$

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \boldsymbol{P}^{-1}\boldsymbol{r}^{(k)}, \tag{6.3}$$

where $\boldsymbol{r}^{(k)}$ denotes the residual after step $k$.

Convergence of stationary methods is readily established. Let the error in iteration $n$ be defined by

$$\boldsymbol{e}^{(n)} = \boldsymbol{x}^{*} - \boldsymbol{x}^{(n)} \tag{6.4}$$

where $\boldsymbol{x}^{*} = \boldsymbol{A}^{-1}b$ is the true solution of Eq. (5.1). The error after the $n$-th iteration $\boldsymbol{e}^{(n)}$ is related to the initial error $\boldsymbol{e}^{(0)}$ by

$$\boldsymbol{e}^{(n)} = (\boldsymbol{I} - \boldsymbol{P}^{-1}\boldsymbol{A})^{n}\boldsymbol{e}^{(0)}. \tag{6.5}$$

It follows immediately that iteration Eq. (6.1) converges if for any consistent matrix norm $\|\|\boldsymbol{I} - \boldsymbol{P}^{-1}\boldsymbol{A}\|\| < 1$. Necessary and sufficient for convergence is that $\rho(\boldsymbol{I} - \boldsymbol{P}^{-1}\boldsymbol{A}) < 1$ holds for the spectral radius (see e.g. [36, Theorem 2.1.1]).

Ideally, termination criteria for iterative methods should be based on the true error in the current iterate and on whether any progress can still be made. Because the true error is generally not available, estimates that are based on the residual and/or on the size of the update steps are often employed. Advantages and disadvantages of different termination criteria are discussed extensively in [9, Section 4.2].

In the algorithms given for the iterative solvers the termination criterion will be specified in general as $\textsc{TermCrit}(\boldsymbol{A}, \boldsymbol{r}, \boldsymbol{b})$ and the iteration be terminated if $\textsc{TermCrit}(\boldsymbol{A}, \boldsymbol{r}^{(k)}, \boldsymbol{b}) < \tau$. The termination criterion that is mostly employed in this thesis is based on the relative residual $\|r^{(k)}\|/\|b\|$. The iteration is stopped when

$$\|r^{(k)}\| \leq \tau\|b\| \tag{6.6}$$

where $\tau$ is the stop tolerance. This criterion, while being sometimes hard to fulfil, has the advantage of not requiring the norm of the operator $\boldsymbol{A}$, which is often expensive to obtain (see also the discussion of Criterion 2 in [9, Section 4.2.1]). Furthermore, this criterion is implemented also in commercial solvers like those in Matlab®, and thus using this one facilitates performance comparisons with those solvers as then both stopping criteria are the same.

### 6.1.1 Implementation of truncated simple iterations

For an implementation of simple iterations (see e.g. [9, Section 2.2]) that works with tensor representations $\mathbf{x}$, e.g. in a low-rank format like the canonical representation (see Section 4.1.4 on page 54), some modifications need to be made to the standard algorithm. For the modification the details of the tensor format are not relevant, but the following points become important:

- The operator $\boldsymbol{A}$ turns into a sum of operators, i.e. $\mathbf{A} = \sum_{i=1}^{L} \mathbf{A}_i$, where each of the $\mathbf{A}_i$ has tensor product structure[1].
- Additions have to be followed by truncations to keep the rank small, i.e. $\mathbf{x} + \mathbf{y}$ becomes $T_\varepsilon(\mathbf{x} + \mathbf{y})$. As this notation becomes unwieldy for repeated additions, the notation $\mathbf{x} \boxplus \mathbf{y} := T_\varepsilon(\mathbf{x}+\mathbf{y})$ is introduced. The binary operation $\boxplus$ is understood to be left-associative, i.e. $\mathbf{x}\boxplus\mathbf{y}\boxplus\mathbf{z} = (\mathbf{x}\boxplus\mathbf{y})\boxplus\mathbf{z}$.
- Computations of norms and inner products, e.g. in the termination criterion, have to modified in order to work with the tensor representation at hand. For the canonical format this is discussed in Section 4.1.4 on page 57.

[1] Note that in the following to distinguish between vectors and matrices on the one hand and tensors and tensor operators on the other, bold italic symbols (e.g. $\boldsymbol{x}$, $\boldsymbol{A}$) are used for the former, while the latter will be displayed in bold upright symbols (e.g. $\mathbf{x}$, $\mathbf{A}$).

For the simple iterations mainly Eq. (6.2) and Eq. (6.3) have to be modified, leading to the perturbed iterative process

$$\begin{aligned}\mathbf{r}^{(k)} &= \mathbf{b} - T_{\mathrm{op}}(T_{\mathrm{op}}(\cdots T_{\mathrm{op}}(\mathbf{A}_1\mathbf{x}^{(k)} + \mathbf{A}_2\mathbf{x}^{(k)})\cdots) + \mathbf{A}_L\mathbf{x}^{(k)}) \\ &= \mathbf{b} - (\mathbf{A}_1\mathbf{x}^{(k)} \boxplus \mathbf{A}_2\mathbf{x}^{(k)} \boxplus \cdots \boxplus \mathbf{A}_L\mathbf{x}^{(k)}) \qquad (6.7)\\ \mathbf{x}^{(k+1)} &= T_{\mathrm{a}}(\mathbf{x}^{(k)} + \mathbf{P}^{-1}T_{\mathrm{b}}(\mathbf{r}^{(k)})). \qquad (6.8)\end{aligned}$$

Since not all truncations have to be performed with the same truncation parameter $\varepsilon$, distinct symbols have been assigned depending on where the truncation is performed. $T_{\mathrm{op}}$ denotes truncation inside the application of the operator $\mathbf{A}$ using truncation parameter $\varepsilon_{\mathrm{op}}$[2], $T_{\mathrm{b}}$ denotes truncation before application of the preconditioner $\mathbf{P}$ using truncation parameter $\varepsilon_b$, and $T_{\mathrm{a}}$ denotes truncation after preconditioner application using truncation parameter $\varepsilon_a$.

The modified algorithm is shown in Alg. 6.1. The tensor truncations are performed in line 6 for the operator, line 8 for the residual (i.e. before preconditioning) and line 13 for the solution (i.e. after preconditioning). A justification, why the operator application in lines 5–7 is performed in the reverse direction, will be given in the following section. Furthermore, the combined effects of the truncations and how the error estimates of Section 5.2 can be applied to the modified algorithm will also be discussed there.

A reference implementation of the algorithm can be found in sglib in the file `solver/generalised_solve_simple`. Since the solver can work on standard full vectors as well as on low-rank formats, it is referred to as "generalised simple iterations" in this work, or gsi for short.

### 6.1.2 Analysis of perturbations

The theorems on error estimates and convergence of perturbed sequences, derived in Section 5.2.1, apply only if the total fixed or relative perturbation for each step in the iterative process

[2]The truncated addition operator $\boxplus$ also refers to $\varepsilon_{\mathrm{op}}$, i.e. in the following we will always assume $\mathbf{x} \boxplus \mathbf{y} := T_{\varepsilon_{\mathrm{op}}}(\mathbf{x} + \mathbf{y})$.

**Algorithm 6.1** Low-rank simple iterations

**Input:** Operator sum $\mathbf{A} = \sum_{i=1}^{L} \mathbf{A}_i$, preconditioner $\mathbf{P}$, right hand side in tensor format $\mathbf{b}$, real number $\tau > 0$

**Output:** vector $\mathbf{x}$ approximates $\mathbf{A}^{-1}\mathbf{b}$

$k \leftarrow 0$
$\mathbf{x}^{(k)} \leftarrow \mathbf{0}$
**for** $k \leftarrow 1, 2, \ldots$ **do**
  $\mathbf{q} \leftarrow \mathbf{0}$
  **for** $i \leftarrow L, L-1, \ldots, 2, 1$ **do**
    $\mathbf{q} \leftarrow T_{\mathrm{op}}(\mathbf{q} + \mathbf{A}_i \mathbf{x}^{(k)})$
  **end for**
  $\mathbf{r}^{(k)} \leftarrow T_{\mathrm{b}}(\mathbf{b} - \mathbf{q})$
  **if** TermCrit$(\mathbf{A}, \mathbf{r}^{(k)}, \mathbf{b})$ **then**
    **exit for**
  **end if**
  $\mathbf{z}^{(k)} \leftarrow \mathbf{P}^{-1}\mathbf{r}^{(k)}$
  $\mathbf{x}^{(k+1)} \leftarrow T_{\mathrm{a}}(\mathbf{x}^{(k)} + \mathbf{z}^{(k)})$
  $k \leftarrow k + 1$
**end for**
$\mathbf{x} \leftarrow \mathbf{x}^{(k)}$

can be bounded. If the truncations are performed as described above with three truncation operators $T_{\mathrm{op}}$, $T_{\mathrm{b}}$ and $T_{\mathrm{a}}$, it is not obvious what the total perturbation parameter $\varepsilon$ in the perturbed update step $\Phi_\varepsilon$ will be. Only in the case that $T_{\mathrm{op}} = T_{\mathrm{b}} = \mathrm{Id}$ and $T_{\mathrm{a}} = T_{\varepsilon_a}$ it is immediately clear that $\varepsilon = \varepsilon_a$. In the other cases, we need to estimate first, how the truncation errors accumulate during multiple additions and truncations, and second, how the application of the preconditioner amplifies or dampens the truncation error. This shall be answered in the following theorems.

For fixed perturbations and multiple truncated additions the following result holds.

**Theorem 6.1.** *Let $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_L$ be a set of vectors and $T_\varepsilon$ a trun-*

*cation operator with fixed truncation such that* $\|T_\varepsilon(\boldsymbol{y}) - \boldsymbol{y}\| \leq \varepsilon$. *For all* $k \geq 0$ *let* $\boldsymbol{s}_k$ *denote the partial sums defined recursively by* $\boldsymbol{s}_0 = 0$ *and* $\boldsymbol{s}_k = \boldsymbol{s}_{k-1} + \boldsymbol{y}_k$. *Let* $\tilde{\boldsymbol{s}}_k$ *denote the truncated partial sums defined by* $\tilde{\boldsymbol{s}}_0 = 0$ *and* $\tilde{\boldsymbol{s}}_k = T_\varepsilon(\tilde{\boldsymbol{s}}_{k-1} + \boldsymbol{y}_k)$. *Then it holds that*

$$\|\tilde{\boldsymbol{s}}_L - \boldsymbol{s}_L\| \leq L\varepsilon. \tag{6.9}$$

*Proof.* With $\boldsymbol{\delta}_k = T_\varepsilon(\tilde{\boldsymbol{s}}_{k-1} + \boldsymbol{y}_k) - (\tilde{\boldsymbol{s}}_{k-1} + \boldsymbol{y}_k)$ and $\|\boldsymbol{\delta}_k\| \leq \varepsilon$ it follows

$$\begin{aligned}\|\tilde{\boldsymbol{s}}_k - \boldsymbol{s}_k\| &= \|(\tilde{\boldsymbol{s}}_{k-1} + \boldsymbol{y}_k + \boldsymbol{\delta}_k) - (\boldsymbol{s}_{k-1} + \boldsymbol{y}_k)\| \\ &\leq \|\tilde{\boldsymbol{s}}_{k-1} - \boldsymbol{s}_{k-1}\| + \|\boldsymbol{\delta}_k\| \\ &\leq \|\tilde{\boldsymbol{s}}_{k-1} - \boldsymbol{s}_{k-1}\| + \varepsilon\end{aligned}$$

the result follows immediately by induction on $\|\tilde{\boldsymbol{s}}_0 - \boldsymbol{s}_0\| = 0$. □

For relative perturbations and multiple truncated additions the following result holds.

**Theorem 6.2.** *Let the assumptions be as in Theorem 6.1, except that* $\|T_\varepsilon(\boldsymbol{y}) - \boldsymbol{y}\| \leq \varepsilon\|\boldsymbol{y}\|$ *holds for the truncation operator. Then it holds that*

$$\|\tilde{\boldsymbol{s}}_L - \boldsymbol{s}_L\| \leq \varepsilon \sum_{i=1}^{L} (1+\varepsilon)^{L-i} \|\boldsymbol{s}_i\|. \tag{6.10}$$

*and*

$$\|\tilde{\boldsymbol{s}}_L - \boldsymbol{s}_L\| \leq \varepsilon(1+\varepsilon)^{L-1}(L\|\boldsymbol{y}_1\| + \cdots + 2\|\boldsymbol{y}_{L-1}\| + \|\boldsymbol{y}_L\|). \tag{6.11}$$

*Proof.* In the same manner as in the previous proof it can be seen that

$$\|\tilde{\boldsymbol{s}}_k - \boldsymbol{s}_k\| \leq \|\tilde{\boldsymbol{s}}_{k-1} - \boldsymbol{s}_{k-1}\| + \varepsilon\|\tilde{\boldsymbol{s}}_{k-1} + \boldsymbol{y}_k\|$$

holds for the difference between the perturbed and the unperturbed partial sums. Using the triangle inequality in the last term

leads to

$$\begin{aligned}\|\tilde{\boldsymbol{s}}_{k-1} + \boldsymbol{y}_k\| &\leq \|\tilde{\boldsymbol{s}}_{k-1} - \boldsymbol{s}_{k-1}\| + \|\boldsymbol{s}_{k-1} + \boldsymbol{y}_k\| \\ &= \|\tilde{\boldsymbol{s}}_{k-1} - \boldsymbol{s}_{k-1}\| + \|\boldsymbol{s}_k\|,\end{aligned}$$

and thus

$$\|\tilde{\boldsymbol{s}}_k - \boldsymbol{s}_k\| \leq (1+\varepsilon)\|\tilde{\boldsymbol{s}}_{k-1} - \boldsymbol{s}_{k-1}\| + \varepsilon\|\boldsymbol{s}_k\|.$$

The first inequality then follows immediately by induction. Noting that $(1+\varepsilon)^{L-i} \leq (1+\varepsilon)^{L-1}$ and $\|\boldsymbol{s}_i\| \leq \|\boldsymbol{y}_1\| + \cdots + \|\boldsymbol{y}_i\|$ for all $i \geq 1$ proves the second inequality. □

Consequently, in order to minimise the overall truncation error, it is advisable to begin adding and truncating with the smaller components and to end with the larger ones. As the norm of the sequence of operators $\mathbf{A}_i$ is in general decreasing—remember that $\sum_{i=1}^{L} \mathbf{A}_i$ is the truncation of a norm convergent infinite sum—it makes sense to reverse the order of summation in Eq. (6.7) and compute the residual as

$$\tilde{\mathbf{r}} = \mathbf{b} - (\mathbf{A}_L\mathbf{x} \boxplus \mathbf{A}_{L-1}\mathbf{x} \boxplus \cdots \boxplus \mathbf{A}_1\mathbf{x}). \tag{6.12}$$

Then it follows using Theorem 6.2 that

$$\|\tilde{\mathbf{r}} - \mathbf{r}\| \leq \varepsilon_{\mathrm{op}}(1+\varepsilon_{\mathrm{op}})^{L-1}(|||\mathbf{A}_1||| + \cdots + L\,|||\mathbf{A}_L|||)\|\mathbf{x}\|, \tag{6.13}$$

where $\mathbf{r}$ is the exact residual without truncations. So the combined relative truncation parameter is given by

$$\tilde{\varepsilon}_{\mathrm{op}} = \varepsilon_{\mathrm{op}}(1+\varepsilon_{\mathrm{op}})^{L-1}(|||\mathbf{A}_1||| + \cdots + L\,|||\mathbf{A}_L|||). \tag{6.14}$$

Finally, the influence of the truncation $T_{\mathrm{b}}$ needs to be estimated. From

$$|||\mathbf{P}^{-1}\mathbf{r} - \mathbf{P}^{-1}T_{\mathrm{b}}(\mathbf{r})||| \leq |||\mathbf{P}^{-1}|||\;|||\mathbf{r} - T_{\mathrm{b}}(\mathbf{r})||| \tag{6.15}$$

follows immediately for the effective truncation parameter $\tilde{\varepsilon}_b$

$$\tilde{\varepsilon}_b = \varepsilon_b |||\mathbf{P}^{-1}|||, \tag{6.16}$$

independent of whether fixed or relative truncations are used. Combining the bounds given above, one can estimate the total truncation per iteration by

$$\begin{aligned}\|\Phi_\varepsilon(\mathbf{x}) - \Phi(\mathbf{x})\| \leq & \tilde{\varepsilon}_{\mathrm{op}} |||\mathbf{P}^{-1}||| \|\mathbf{x}\| + \\ & (\varepsilon_a + \tilde{\varepsilon}_b)(\|\mathbf{b}\| + |||\mathbf{A}||| \, \|\mathbf{x}\|),\end{aligned} \tag{6.17}$$

where relative perturbations have been assumed everywhere. Note that while this gives an upper bound for the induced perturbation, it often overestimates the true perturbation often considerably. This will be shown in the following example.

**Example 6.3.** *In the following estimates computed by Eq.* (6.14) *and Eq.* (6.16) *will be given separately and compared to actual perturbations computed numerically. The model used is the small model of Section 8.1.1, in order to make computations of operator norms feasible. Only relative truncations were used with truncation parameters set to* $\varepsilon_{\mathrm{op}} = \varepsilon_b = 10^{-4}$. *Further, the tensor* $\mathbf{x}$ *was set equal to the right hand side* $\mathbf{f}$, *as this is a common starting guess in iterative methods. Numbers are always rounded to two decimal places. The code for this example can be found in* sglib *in the script* `tests/test_estimate_truncations.`

*For the given parameters and model Eq.* (6.14) *results in the estimate* $\|\tilde{\mathbf{r}} - \mathbf{r}\| \leq 0.21$, *while direct comparison shows* $\|\tilde{\mathbf{r}} - \mathbf{r}\| = 0.0015$, *i.e. the perturbation is overestimated by a factor of about* 140. *Just for comparison the operator has also been evaluated in normal order, resulting in an estimate of* $\|\tilde{\mathbf{r}} - \mathbf{r}\| \leq 0.48$ *and a computed difference shows* $\|\tilde{\mathbf{r}} - \mathbf{r}\| = 0.0032$ *with a factor of about* 150. *This shows that Eq.* (6.14) *gives qualitatively the correct results, and that indeed reverse summing of the operator leads to smaller perturbation errors.*

*For truncation before preconditioning Eq.* (6.16) *results in an estimate* $|||\mathbf{P}^{-1}\mathbf{r} - \mathbf{P}^{-1}T_{\mathrm{b}}(\mathbf{r})||| \leq 0.18$ *while computation produced*

$|||\mathbf{P}^{-1}\mathbf{r} - \mathbf{P}^{-1}T_{\mathrm{b}}(\mathbf{r})||| = 0.010$. *Here, the perturbation error is still overestimated, albeit with a smaller factor of about* 18.

Note that though the truncations in Eq. (6.17) are all of the relative type, Eq. (6.17) itself is neither a fixed nor a relative truncation, but a combination of both. The equation in fact has the form

$$\|\Phi_\varepsilon(\mathbf{x}) - \Phi(\mathbf{x})\| \leq \varepsilon_f + \varepsilon_r\|\boldsymbol{x}\| \tag{6.18}$$

with fixed truncation parameter $\varepsilon_f = (\varepsilon_a + \tilde{\varepsilon}_b)\|\mathbf{b}\|$ and relative truncation parameter $\varepsilon_r = \tilde{\varepsilon}_{\mathrm{op}}|||\mathbf{P}^{-1}||| + (\varepsilon_a + \tilde{\varepsilon}_b)\,|||\mathbf{A}|||$. However, the error estimates for the relative perturbations from Section 5.2 can be easily adapted to accommodate this form. The a priori error for iterations of the form Eq. (6.18) becomes

$$\|\tilde{x}^{(k)} - x^*\| \leq \tilde{q}^k\,\|x^{(0)} - x^*\| + (\varepsilon_f + \varepsilon_r c_x)\,\frac{1-\tilde{q}^k}{1-\tilde{q}}. \tag{6.19}$$

with $\tilde{q} = q + \varepsilon_r$. This follows immediately from the proof of Theorem 5.6 when the estimate Eq. (5.22) is changed according to Eq. (6.18). The argument given in the proof of Theorem 5.8 can be used unchanged also for this case leading to the a posteriori error estimate

$$\|\tilde{x}^{(k)} - x^*\| \leq \frac{q}{1-q}\|\tilde{x}^{(k)} - \tilde{x}^{(k-1)}\| + \frac{\varepsilon_f + \varepsilon_r\|\tilde{x}^{(k-1)}\|}{1-q}. \tag{6.20}$$

## 6.2 Conjugate Gradients

In this section some basic facts about Krylov subspace methods in general and the conjugate gradient method (CG) in particular shall be recollected (see e.g. [9, 36, 89]). If the system at hand is symmetric and positive definite the Krylov subspace method of choice is usually the conjugate gradient method, as it does not require long recurrences, is very stable and converges quite fast in general.

Assuming that $\boldsymbol{P}$ is the identity in Eq. (6.1), it is obvious that

the $k$-th iterate is contained in the so-called Krylov subspace

$$\mathcal{K}_k(\boldsymbol{A}, \boldsymbol{r}_0) = \text{span}\{\boldsymbol{r}_0, \boldsymbol{A}\boldsymbol{r}_0, \ldots, \boldsymbol{A}^{k-1}\boldsymbol{r}_0\}, \tag{6.21}$$

or rather in $\boldsymbol{x}_0 + \mathcal{K}_k(\boldsymbol{A}, \boldsymbol{r}_0)$, if the iteration starts with a non-zero initial guess $\boldsymbol{x}_0$. The essence of Krylov subspace methods is to find the best iterate that minimises some norm of the error over $\mathcal{K}_k$ (see e.g. [89]) in every iteration. In case of conjugate gradients this is the energy norm, so that in each step $k$

$$\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^*\|_{\boldsymbol{A}} = \min_{\boldsymbol{x} \in \mathcal{K}_k} \|\boldsymbol{x} - \boldsymbol{x}^*\|_{\boldsymbol{A}} \tag{6.22}$$

holds. It turns out that the update of the iterate $\boldsymbol{x}^{(k)}$ can be written as

$$\alpha^{(k-1)} = \frac{\langle \boldsymbol{r}^{(k-1)} | \boldsymbol{p}^{(k-1)} \rangle}{\langle \boldsymbol{p}^{(k-1)} | \boldsymbol{A}\boldsymbol{p}^{(k-1)} \rangle} \tag{6.23}$$

$$\boldsymbol{x}^{(k)} = \boldsymbol{x}^{(k-1)} + \alpha^{(k-1)} \boldsymbol{p}^{(k-1)}, \tag{6.24}$$

where $\boldsymbol{p}^{(k)}$ is the search direction and $\alpha^{(k-1)}$ the minimiser from the line search for Eq. (6.22). The update of the search directions is done via

$$\beta^{(k-1)} = \frac{\langle \boldsymbol{r}^{(k)} | \boldsymbol{r}^{(k)} \rangle}{\langle \boldsymbol{r}^{(k-1)} | \boldsymbol{r}^{(k-1)} \rangle} \tag{6.25}$$

$$\boldsymbol{p}^{(k)} = \boldsymbol{r}^{(k)} + \beta^{(k-1)} \boldsymbol{p}^{(k-1)}, \tag{6.26}$$

keeping the $\boldsymbol{p}^{(k)}$ mutually conjugate, i.e. $\boldsymbol{A}$-orthogonal.

The conjugate gradient method can be preconditioned if the preconditioner $\boldsymbol{P}$ is symmetric positive-definite. Fortunately, it is not necessary to compute the Cholesky decomposition of $\boldsymbol{P}$ and use symmetric preconditioning [9]. Rather the algorithm can be modified to employ the preconditioned residual $\boldsymbol{z}^{(k)} = \boldsymbol{P}^{-1}\boldsymbol{r}^{(k)}$ by modifying Eq. (6.23), Eq. (6.25), and Eq. (6.26) as follows:

$$\alpha^{(k-1)} = \frac{\langle \boldsymbol{r}^{(k-1)} | \boldsymbol{z}^{(k-1)} \rangle}{\langle \boldsymbol{p}^{(k-1)} | \boldsymbol{A}\boldsymbol{p}^{(k-1)} \rangle},$$

$$\beta^{(k-1)} = \frac{\langle \boldsymbol{r}^{(k)} | \boldsymbol{z}^{(k)} \rangle}{\langle \boldsymbol{r}^{(k-1)} | \boldsymbol{z}^{(k-1)} \rangle},$$
$$\boldsymbol{p}^{(z)} = \boldsymbol{r}^{(k)} + \beta^{(k-1)} \boldsymbol{p}^{(k-1)}. \tag{6.27}$$

The resulting algorithm is called the *preconditioned conjugate gradient* method (PCG).

### 6.2.1 Convergence of the CG method

As the CG method has not the simple form as Eq. (6.1) the analysis of the convergence of CG and the effects of perturbations is more involved as for simple iterations. Convergence of the standard CG method is studied in many books on Krylov subspace methods (see e.g. [36, 89]), from which the points relevant for this thesis shall be summarised here. As already pointed out, CG always leads to the best approximation in the Krylov subspace $\mathcal{K}_k$ with respect to the $\boldsymbol{A}$-norm of the error. Since the best approximation in $\mathcal{K}_k$ is a polynomial $p_k$ in $\boldsymbol{A}$, one can get an upper bound for the error via interpolation by Chebyshev polynomials on the interval $[\lambda_{\min}, \lambda_{\max}]$, where $\lambda_{\min}$ and $\lambda_{\max}$ are the minimal and maximal eigenvalue of $\boldsymbol{A}$, respectively. The estimate thus obtained is

$$\frac{\|\boldsymbol{e}^{(k)}\|_{\boldsymbol{A}}}{\|\boldsymbol{e}^{(0)}\|_{\boldsymbol{A}}} \leq 2 \left[ \left( \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1} \right)^k + \left( \frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1} \right)^k \right]^{-1} \leq 2 \left( \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1} \right)^k, \tag{6.28}$$

where $\kappa = \lambda_{\max}/\lambda_{\min}$ is the spectral condition number of $\boldsymbol{A}$ [36]. For the preconditioned conjugate gradient methods $\kappa$ is the spectral condition number of $\boldsymbol{P}^{-1}\boldsymbol{A}$ (see e.g. [36]).

The last estimate in Eq. (6.28) is the commonly reported one for the convergence of CG. It shows that in CG the square root of the condition number enters into the rate of convergence, in contrast to, for example, steepest descent in which the condition number enters directly, making CG asymptotically superior. Due to the factor 2, however, it does not show for a single step of the iteration that the iteration map is contractive as required by the theorems of Section 5.2.

Setting $k = 1$ in the first inequality leads to

$$\frac{\|\boldsymbol{e}^{(1)}\|_{\boldsymbol{A}}}{\|\boldsymbol{e}^{(0)}\|_{\boldsymbol{A}}} \leq \frac{\kappa - 1}{\kappa + 1}. \tag{6.29}$$

This estimate is the same as the convergence rate of steepest descent, which is dependent on the condition number itself instead of its square root. However, this estimate can be used for the contractivity $q$ of the iteration operator and thus allows the a priori estimates in Theorems 5.3 and 5.6 and the a posteriori estimates in Theorems 5.5 and 5.8 to be applied.

In numerical experiments it turned out that for sufficiently small values of the truncation parameters convergence rates like in Eq. (6.28) can be recovered, but often deteriorated for larger truncation parameters to rates like in Eq. (6.29).

### 6.2.2 Implementation of conjugate gradients with truncation

The modifications that need to be done for an implementation of the preconditioned conjugate gradient method to work with low-rank formats are essentially the same as for the simple iterations (see Section 6.1.1). However, there is one point in the standard CG algorithm that needs attention. In the CG algorithm the residual is efficiently computed by the update formula

$$\mathbf{r}^{(k)} = \mathbf{r}^{(k-1)} - \alpha^{(k-1)}\mathbf{A}\mathbf{p}^{(k-1)}, \tag{6.30}$$

because then application of the operator $\mathbf{A}$ needs to be performed only once per iteration. The standard formula

$$\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)} \tag{6.31}$$

needs an extra evaluation, which in standard codes (see e.g. the PCG implementation in Matlab® [60]) is only performed when the iterations stagnate or to "double-check" the termination criterion.

If in codes based on tensor formats the update formula Eq. (6.30) is used, the error in the residual builds up very fast and the residual

becomes very inaccurate after just a few iterations. Because CG is very sensitive to errors in the residual (see e.g. [36, 89]), this leads to severe deterioration of the convergence rate, making it necessary to recompute the residual based on Eq. (6.31) frequently. Since this destroys any performance gain offered by the residual update formula Eq. (6.30), only the standard formula for the residual Eq. (6.31) is used in the low-rank solvers presented here.

A sample algorithm for the modified PCG is given in Alg. 6.2, where the application of the application of the operator $\mathbf{A}$ to the tensor $\mathbf{x}$ with truncations using the truncation operator $T_\varepsilon$ is denoted by $\textsc{ApplyTrunc}(\mathbf{A}, \mathbf{x}, T_\varepsilon)$ for shorter presentation.

A sample implementation in sglib can be found in the file `solver/generalised_solve_pcg`. Since this solver works with standard full vectors as well as with low-rank formats, it is referred to as "generalised PCG" in this work, or `gpcg` for short.

**Algorithm 6.2** Low-rank Preconditioned Conjugate Gradients

**Input:** Operator sum $\mathbf{A} = \sum_{i=1}^{L} \mathbf{A}_i$, preconditioner $\mathbf{P}$, right hand side in low-rank format $\mathbf{b}$, initial approximation $\mathbf{x}^{(0)}$

**Output:** approximation $\mathbf{x}$ in low-rank format for $\mathbf{A}^{-1}\mathbf{b}$

$\mathbf{q} \leftarrow \textsc{ApplyTrunc}(\mathbf{A}, \mathbf{x}^{(0)}, T_{\mathrm{op}})$
$\mathbf{r}^{(0)} \leftarrow T_{\mathrm{b}}(\mathbf{b} - \mathbf{q})$
$\mathbf{z}^{(0)} \leftarrow \mathbf{P}^{-1}\mathbf{r}^{(0)}$ ▷ i.e. solve $\mathbf{P}\mathbf{z}^{(0)} = \mathbf{r}^{(0)}$
$\mathbf{p}^{(0)} \leftarrow \mathbf{z}^{(0)}$
**for** $k \leftarrow 1, 2, \ldots$ **do**
  $\mathbf{q} \leftarrow \textsc{ApplyTrunc}(\mathbf{A}, \mathbf{p}^{(k-1)}, T_{\mathrm{op}})$
  $\alpha^{(k-1)} \leftarrow \frac{\langle \mathbf{r}^{(k-1)} | \mathbf{z}^{(k-1)} \rangle}{\langle \mathbf{p}^{(k-1)} | \mathbf{q} \rangle}$
  $\mathbf{x}^{(k)} \leftarrow \mathbf{x}^{(k-1)} + \alpha^{(k-1)}\mathbf{p}^{(k-1)}$
  $\mathbf{q} \leftarrow \textsc{ApplyTrunc}(\mathbf{A}, \mathbf{x}^{(k)}, T_{\mathrm{op}})$
  $\mathbf{r}^{(k)} \leftarrow T_{\mathrm{b}}(\mathbf{b} - \mathbf{q})$
  **if** $\textsc{TermCrit}(\mathbf{A}, \mathbf{r}^{(k)}, \mathbf{b})$ **then**
    **exit for**
  **end if**
  $\mathbf{z}^{(k)} \leftarrow \mathbf{P}^{-1}\mathbf{r}^{(k)}$ ▷ i.e. solve $\mathbf{P}\mathbf{z}^{(k)} = \mathbf{r}^{(k)}$
  $\beta^{(k-1)} \leftarrow \frac{\langle \mathbf{r}^{(k)} | \mathbf{z}^{(k)} \rangle}{\langle \mathbf{r}^{(k-1)} | \mathbf{z}^{(k-1)} \rangle}$
  $\mathbf{p}^{(k)} \leftarrow \mathbf{z}^{(k)} + \beta^{(k-1)}\mathbf{p}^{(k-1)}$
**end for**
$\mathbf{x} \leftarrow \mathbf{x}^{(k)}$

# Chapter 7

# Preconditioning

*This chapter studies preconditioning of linear systems in tensor form. The first section deals with the case that the operator is given in tensor form, regardless of whether the solution is given in a full or a low-rank format. Most of this is already covered in the literature, but one new preconditioner with favourable properties for simple iterations is presented. The second section deals with techniques that can be used to improve performance of linear solvers, when the solution is represented in a low-rank format.*

## 7.1 Tensor operator preconditioning

Most iterative methods for solving linear systems of the form

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{7.1}$$

can be accelerated when the system is multiplied from the left by an operator $\mathbf{P}^{-1}$ such that $\mathbf{P}^{-1}\mathbf{A}$ is in some sense, which may depend on the iterative method, close to the identity or, equivalently, $\mathbf{P}$ is in some sense close to $\mathbf{A}$. The preconditioned system to be solved by the iterative method is then

$$\mathbf{P}^{-1}\mathbf{A}\mathbf{x} = \mathbf{P}^{-1}\mathbf{b}. \tag{7.2}$$

Furthermore, $\mathbf{P}$ must be efficiently invertible, i.e. solving with $\mathbf{P}$ must be much faster than solving with $\mathbf{A}$. An important fact for

preconditioning the linear operator $\mathbf{A}$ in the tensor equation

$$\mathbf{A}\mathbf{x} = \left( \sum_{i=0}^{L} \boldsymbol{A}_i \otimes \boldsymbol{B}_i \right) \mathbf{x} \tag{7.3}$$

is that elementary tensor products of operators are simple to invert, since $(\boldsymbol{A} \otimes \boldsymbol{B})^{-1} = \boldsymbol{A}^{-1} \otimes \boldsymbol{B}^{-1}$. Most preconditioners for Eq. (7.1) thus have the form

$$\mathbf{P} = \boldsymbol{P} \otimes \boldsymbol{Q}. \tag{7.4}$$

If the tensor $\mathbf{x}$ is e.g. in canonical format $\mathbf{x} = (\boldsymbol{s}; \boldsymbol{X}, \boldsymbol{Y})$ application of the preconditioner gives

$$\mathbf{P}^{-1}\mathbf{x} = \left( \boldsymbol{s}; \boldsymbol{P}^{-1}\boldsymbol{X}, \boldsymbol{Q}^{-1}\boldsymbol{Y} \right). \tag{7.5}$$

Applied to a tensor in matrix format $\boldsymbol{X}$ gives

$$\mathbf{P}^{-1}\boldsymbol{X} = \boldsymbol{P}^{-1}\boldsymbol{X}\boldsymbol{Q}^{-T} = (\boldsymbol{Q}^{-1}(\boldsymbol{P}^{-1}\boldsymbol{X})^T)^T. \tag{7.6}$$

A preconditioner like $\mathbf{P} = \boldsymbol{P}_1 \otimes \boldsymbol{Q}_1 + \boldsymbol{P}_2 \otimes \boldsymbol{Q}_2$ would not lead to equally simple formulas and would be nearly as difficult to solve with as Eq. (7.1) itself.[1]

### 7.1.1 Mean based preconditioner

One common preconditioner of this form is the so called *mean based preconditioner* [27, 74, 77, 85]. For stochastic systems, especially when the standard deviation of the conductivity is relatively small compared to its mean, the deterministic part of the operator $\boldsymbol{K}_0 \otimes \boldsymbol{\Delta}_0$ is dominating the operator in Eq. (3.24). In the general setting here this would correspond to using

$$\mathbf{P}_{\mathsf{mean}} = \boldsymbol{A}_0 \otimes \boldsymbol{B}_0 \tag{7.7}$$

[1] Note that also preconditioners of the form $\mathbf{P}^{-1} = \boldsymbol{P}_1^{-1} \otimes \boldsymbol{Q}_1^{-1} + \boldsymbol{P}_2^{-1} \otimes \boldsymbol{Q}_2^{-1}$ could be applied efficiently. However, to the author's knowledge no results on this type of preconditioner have been reported in the literature so far.

as the preconditioner. However, as this corresponds in the stochastic setting to $\mathbf{P}_{\mathsf{mean}} = \boldsymbol{K}_0 \otimes \boldsymbol{\Delta}_0$ it is generally called the *mean based preconditioner*. However, Eq. (7.7) can also be used for general tensor equations as long as $\boldsymbol{A}_0 \otimes \boldsymbol{B}_0$ dominates the operator in some sense.

Advantages of this preconditioner are that it is readily available from the operator $\mathbf{A}$ without any computational overhead, and further that, if $\mathbf{A}$ corresponds to a positive definite elliptic SPDE, then $\mathbf{P}_{\mathsf{mean}}$ is also symmetric and positive definite, which is especially important for the CG method. If the spatial dimension is not too large, as it is mostly here in this work, the full LU or Cholesky decomposition of $\boldsymbol{A}_0$ and $\boldsymbol{B}_0$ can be computed, where the latter is mostly diagonal anyway. In some works also the incomplete LU or Cholesky decomposition are used or algebraic multigrid methods (AMG), which, however, do not seem to perform as well as the exact decomposition [77].

### 7.1.2 Kronecker product preconditioner

A preconditioner that has better spectral properties especially for the conjugate gradient method was proposed by Ullmann [85] based on work of van Loan and Pitsianis [88]. Given some fixed matrix $\boldsymbol{P}$ a best approximation for $\mathbf{A}$ in the form $\boldsymbol{P} \otimes \boldsymbol{Q}$ is constructed by solving the minimisation problem

$$\boldsymbol{Q} = \arg\min_{\boldsymbol{Q}'} \left| \left| \left| \mathbf{A} - \boldsymbol{P} \otimes \boldsymbol{Q}' \right| \right| \right| . \tag{7.8}$$

for $\boldsymbol{Q}$. With most norms this is a very difficult problem to solve. In the Frobenius norm $|||\boldsymbol{A}|||_F$, however, which is defined via the scalar product $\langle \boldsymbol{A} | \boldsymbol{B} \rangle_F = \mathrm{Tr}(\boldsymbol{A}^T \boldsymbol{B})$, it has a simple analytical solution. Using the relation

$$\frac{\partial \langle \boldsymbol{A} | \boldsymbol{X} \rangle_F}{\partial \boldsymbol{X}} = \boldsymbol{A} \tag{7.9}$$

(see e.g. [75, Section 2.5.1]) one obtains for the minimiser

$$\boldsymbol{Q} = \sum_{i=0}^{L} \frac{\langle \boldsymbol{P} | \boldsymbol{A}_i \rangle_F}{\langle \boldsymbol{P} | \boldsymbol{P} \rangle_F} \boldsymbol{B}_i \tag{7.10}$$

(see [88]), which can be evaluated in a numerically efficient way. Setting $\boldsymbol{P} = \boldsymbol{A}_0$ in Eq. (7.10) leads to

$$\mathbf{P}_{\mathsf{kron}} = \boldsymbol{A}_0 \otimes \left( \sum_{i=0}^{L} \frac{\langle \boldsymbol{A}_0 | \boldsymbol{A}_i \rangle_F}{\langle \boldsymbol{A}_0 | \boldsymbol{A}_0 \rangle_F} \boldsymbol{B}_i \right), \tag{7.11}$$

which is known as the *Kronecker product preconditioner* This preconditioner was used by Ullmann [85] and Powell and Ullmann [77]. Like the mean based preconditioner, the Kronecker product preconditioner $\mathbf{P}_{\mathsf{kron}}$ is also symmetric positive definite if the operator $\mathbf{A}$ is.

**Remark 7.1.** *Note that the minimisation in Eq.* (7.8) *can be performed iteratively for* $\boldsymbol{Q}$ *and* $\boldsymbol{P}$ *in turn. According to [88] this iteration converges to the minimiser over both* $\boldsymbol{Q}$ *and* $\boldsymbol{P}$*. Due to the symmetry of the tensor product for the minimisation of* $\boldsymbol{P}$ *Eq.* (7.10) *can also be applied with the roles of the matrices* $\boldsymbol{Q}$ *and* $\boldsymbol{P}$ *and those of* $\boldsymbol{A}_i$ *and* $\boldsymbol{B}_i$ *interchanged. A preconditioner* $\mathbf{P}_{\mathsf{kron\text{-}3}}$*, where this iteration has been performed 3 times, has also been tested numerically (see Table 7.1 and the discussion thereafter).*

### 7.1.3 Inverse Kronecker product preconditioner

While the Kronecker product preconditioner was shown to perform well for CG on symmetric positive definite systems by Ullmann [85] and for MINRES on symmetric saddle point systems by Powell and Ullmann [77], it did not work so well on many examples studied in this work. Especially for simple iterations the use of $\mathbf{P}_{\mathsf{kron}}$ sometimes lead to divergent iterations.

The rate of convergence of simple iterations is determined by the spectral radius of the iteration operator $\mathbf{I} - \mathbf{P}^{-1}\mathbf{A}$ (see Section 6.1). However, the minimisation of $|||\mathbf{P} - \mathbf{A}|||_F$, which is done for the

Kronecker product preconditioner, does not necessarily lead to a small spectral radius of $\mathbf{I}-\mathbf{P}^{-1}\mathbf{A}$, and can even increase it in some cases. Since solving the minimisation problem for the spectral radius is a very difficult nonlinear problem, we can instead (in analogy to Eq. (7.8)) try and solve the minimisation problem

$$\boldsymbol{V} = \arg\min_{\boldsymbol{V}'} \left|\left|\left|\mathbf{I} - (\boldsymbol{U}\otimes\boldsymbol{V}')\mathbf{A}\right|\right|\right|_F \tag{7.12}$$

in the Frobenius norm, where $\boldsymbol{U} = \boldsymbol{P}^{-1}$. Using the equalities

$$\frac{\partial\langle \boldsymbol{I}|\boldsymbol{X}\boldsymbol{A}\rangle_F}{\partial \boldsymbol{X}} = \frac{\partial\langle \boldsymbol{A}^T|\boldsymbol{X}\rangle_F}{\partial \boldsymbol{X}} = \boldsymbol{A}^T \tag{7.13}$$

$$\frac{\partial\langle \boldsymbol{X}\boldsymbol{A}|\boldsymbol{X}\boldsymbol{B}\rangle_F}{\partial \boldsymbol{X}} = \boldsymbol{X}(\boldsymbol{A}\boldsymbol{B}^T + \boldsymbol{B}\boldsymbol{A}^T). \tag{7.14}$$

for derivatives of the Frobenius inner product (see e.g. [75, Section 2.5.2]) the problem can be solved analytically:

$$\begin{aligned}
\boldsymbol{V} &= \arg\min_{\boldsymbol{V}'} \left|\left|\left|\mathbf{I} - (\boldsymbol{U}\otimes\boldsymbol{V}')\mathbf{A}\right|\right|\right|_F \\
&= \arg\min_{\boldsymbol{V}'} \Bigg( \sum_{i,j} \langle \boldsymbol{U}\boldsymbol{A}_i|\boldsymbol{U}\boldsymbol{A}_j\rangle_F \langle \boldsymbol{V}'\boldsymbol{B}_i|\boldsymbol{V}'\boldsymbol{B}_j\rangle_F \\
&\qquad\qquad - 2\sum_i \langle \boldsymbol{I}|\boldsymbol{U}\boldsymbol{A}_i\rangle_F \langle \boldsymbol{I}|\boldsymbol{V}'\boldsymbol{B}_i\rangle_F \Bigg) \\
&= \operatorname*{root}_{\boldsymbol{V}'} \left( \boldsymbol{V}' \sum_{i,j} \langle \boldsymbol{U}\boldsymbol{A}_i|\boldsymbol{U}\boldsymbol{A}_j\rangle_F \boldsymbol{B}_i^T\boldsymbol{B}_j - \sum_i \langle \boldsymbol{I}|\boldsymbol{U}\boldsymbol{A}_i\rangle_F \boldsymbol{B}_i^T \right) \\
&= \underbrace{\left( \sum_i \langle \boldsymbol{I}|\boldsymbol{U}\boldsymbol{A}_i\rangle_F \boldsymbol{B}_i^T \right)}_{\boldsymbol{K}} \underbrace{\left( \sum_{i,j} \langle \boldsymbol{U}\boldsymbol{A}_i|\boldsymbol{U}\boldsymbol{A}_j\rangle_F \boldsymbol{B}_i^T\boldsymbol{B}_j \right)^{-1}}_{\boldsymbol{H}}
\end{aligned} \tag{7.15}$$

The preconditioner, which will be called the *inverse Kronecker product preconditioner* in this thesis, can then be written as

$$\mathbf{P}_{\mathsf{ikron}} = \boldsymbol{U}^{-1}\otimes\boldsymbol{V}^{-1} = \boldsymbol{P}\otimes(\boldsymbol{H}\boldsymbol{K}^{-1}),$$

with (as for the Kronecker product preconditioner) $\boldsymbol{P} = \boldsymbol{A}_0$ and

$$\boldsymbol{K} = \sum_i \langle \boldsymbol{I} | \boldsymbol{A}_0^{-1} \boldsymbol{A}_i \rangle_F \boldsymbol{B}_i^T \tag{7.16}$$

$$\boldsymbol{H} = \sum_{i,j} \langle \boldsymbol{A}_0^{-1} \boldsymbol{A}_i | \boldsymbol{A}_0^{-1} \boldsymbol{A}_j \rangle_F \boldsymbol{B}_i^T \boldsymbol{B}_j. \tag{7.17}$$

One obvious drawback of this preconditioner is that it is in general not symmetric. It is hence unusable for methods like CG, which requires the preconditioner to be symmetric positive definite. An implementation of this preconditioner and the preconditioners of the preceding sections can be found in sglib in the file `stochastic_precond_mean_based`.

| Preconditioner | $\mathbf{P}_{\text{mean}}$ | $\mathbf{P}_{\text{kron}}$ | $\mathbf{P}_{\text{kron-3}}$ | $\mathbf{P}_{\text{ikron}}$ |
|---|---|---|---|---|
| Setup time (s) | 0.0024 | 0.0057 | 0.0092 | 12.20 |
| $\rho(\mathbf{P}-\mathbf{A})$ | 14.28 | 6.92 | 6.88 | 8.46 |
| $\lvert\lvert\lvert\mathbf{P}-\mathbf{A}\rvert\rvert\rvert_2$ | 14.28 | 6.92 | 6.88 | 8.52 |
| $\lvert\lvert\lvert\mathbf{P}-\mathbf{A}\rvert\rvert\rvert_F$ | 353.48 | 261.69 | 261.55 | 275.93 |
| $\rho(\mathbf{I}-\mathbf{P}^{-1}\mathbf{A})$ | 0.900 | 0.973 | 0.969 | 0.894 |
| $\left\lvert\left\lvert\left\lvert\mathbf{I}-\mathbf{P}^{-1}\mathbf{A}\right\rvert\right\rvert\right\rvert_2$ | 0.967 | 1.088 | 1.082 | 0.937 |
| $\left\lvert\left\lvert\left\lvert\mathbf{I}-\mathbf{P}^{-1}\mathbf{A}\right\rvert\right\rvert\right\rvert_F$ | 77.31 | 70.68 | 70.66 | 66.75 |
| $n_{\text{pcg}}$ | 23 | 25 | 24 | — |
| $n_{\text{gsi}}$ | 81 | $> 300$ | 280 | 76 |

**Table 7.1:** Comparison of different measures of similarity (i.e. the Frobenius norm, the spectral norm and the spectral radius) for the stochastics preconditioners $\mathbf{P}_{\text{mean}}$, $\mathbf{P}_{\text{kron}}$, $\mathbf{P}_{\text{ikron}}$ and $\mathbf{P}_{\text{kron-3}}$. In the last two rows are the number of iterations needed for a conjugate gradient (pcg) and a simple iteration solver (gsi) for solving the medium model using the given preconditioner (see Section 8.1). (Script: `tests/test_preconditioners`)

In order to compare the preconditioners of this and the preceding sections some numerical experiments have been conducted which are summarised in Table 7.1. The main goal was to measure how much the norms and spectral radii of $\mathbf{P}-\mathbf{A}$ and $\mathbf{I}-\mathbf{P}^{-1}\mathbf{A}$

are indeed reduced and to compare the actual performance of the preconditioners on a model problem. The following observations can be made from Table 7.1:

- The Frobenius norm of $\mathbf{P} - \mathbf{A}$ is smallest for $\mathbf{P}_{\mathsf{kron}}$ and $\mathbf{P}_{\mathsf{kron\text{-}3}}$ with $\mathbf{P}_{\mathsf{kron\text{-}3}}$ being only minimally smaller. The same holds also for the spectral norm and the spectral radius, which coincide for each preconditioner except $\mathbf{P}_{\mathsf{ikron}}$, since $\mathbf{P} - \mathbf{A}$ is symmetric.

- The Frobenius norm of $\mathbf{I} - \mathbf{P}^{-1}\mathbf{A}$ is smallest for the inverse Kronecker product preconditioner $\mathbf{P}_{\mathsf{ikron}}$. For $\mathbf{P}_{\mathsf{kron}}$ and $\mathbf{P}_{\mathsf{kron\text{-}3}}$ the Frobenius norm is smaller than for the mean based preconditioner $\mathbf{P}_{\mathsf{mean}}$[2].

- The spectral norm and spectral radius of $\mathbf{I} - \mathbf{P}^{-1}\mathbf{A}$ are higher for $\mathbf{P}_{\mathsf{kron}}$ and $\mathbf{P}_{\mathsf{kron\text{-}3}}$ than for $\mathbf{P}_{\mathsf{ikron}}$ and $\mathbf{P}_{\mathsf{mean}}$. This difference is especially significant, when considering that the speed of convergence for simple iterations is asymptotically proportional to $1/\log(\rho)$ (see e.g. [89]), which can be well approximated by $1/(1-\rho)$ for $\rho \approx 1$.

- The number of iterations for PCG is roughly the same for all preconditioners. The number of iterations is larger for $\mathbf{P}_{\mathsf{kron}}$ and $\mathbf{P}_{\mathsf{kron\text{-}3}}$ than for $\mathbf{P}_{\mathsf{mean}}$, contrary to what was expected. Note that $\mathbf{P}_{\mathsf{ikron}}$ cannot be used for PCG as it is not symmetric.

- The number of iterations for simple iterations is smallest for $\mathbf{P}_{\mathsf{ikron}}$ and $\mathbf{P}_{\mathsf{mean}}$. Convergence is very slow for $\mathbf{P}_{\mathsf{kron\text{-}3}}$, and the solver did not converge for $\mathbf{P}_{\mathsf{kron}}$ within 300 iterations.

- The setup times $\mathbf{P}_{\mathsf{mean}}$, $\mathbf{P}_{\mathsf{kron}}$ and $\mathbf{P}_{\mathsf{kron\text{-}3}}$ are very small compared to the total runtime spent in the solvers. In contrast, the setup time for $\mathbf{P}_{\mathsf{ikron}}$ is prohibitively large,

[2] However, this was often not the case in other numerical experiments conducted by the author, in which the Frobenius norm of $\mathbf{I} - \mathbf{P}^{-1}\mathbf{A}$ was often even larger for $\mathbf{P}_{\mathsf{kron}}$ and $\mathbf{P}_{\mathsf{kron\text{-}3}}$ than for $\mathbf{P}_{\mathsf{mean}}$.

being approximately three orders of magnitude slower than for e.g. $\mathbf{P}_{\text{kron-3}}$.

In summary, the Kronecker product preconditioners $\mathbf{P}_{\text{kron}}$ and $\mathbf{P}_{\text{kron-3}}$ did not perform better for the PCG solver than $\mathbf{P}_{\text{mean}}$ and $\mathbf{P}_{\text{ikron}}$, and much worse for the simple iterations. While the inverse Kronecker product performed slightly better there, the gain is relatively small. The cost of constructing this preconditioner currently outweighs the savings in iterations. Thus, in the following only the mean based preconditioner $\mathbf{P}_{\text{mean}}$ has been used, since it showed the best relation between performance and cost of construction and application.

## 7.2 Preconditioning strategies

When examining the update step of the simple iterative method

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{P}^{-1}(\mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}) \tag{7.18}$$

the most performance-critical part is the evaluation of the operator on the right hand side and the truncations occurring inside. The operator has the following form

$$\mathbf{A}\mathbf{x} = \sum_{i=1}^{L} \mathbf{A}_i \mathbf{x}. \tag{7.19}$$

Performing the additions with truncation leads to

$$\mathbf{A}\mathbf{x} \approx \mathbf{A}_1\mathbf{x} \boxplus \mathbf{A}_2\mathbf{x} \boxplus \cdots \boxplus \mathbf{A}_L\mathbf{x}, \tag{7.20}$$

where the symbol $\boxplus$ denotes addition with subsequent truncation and the expression is evaluated from left to right.

Now, in Eq. (7.18) it is rather the term $\mathbf{P}^{-1}\mathbf{A}\mathbf{x}$ that needs to be evaluated. This, however, can be done in two forms. The first one

$$\mathbf{P}^{-1}\mathbf{A}\mathbf{x} \approx \mathbf{P}^{-1}(\mathbf{A}_1\mathbf{x} \boxplus \mathbf{A}_2\mathbf{x} \boxplus \cdots \boxplus \mathbf{A}_L\mathbf{x}), \tag{7.21}$$

is common, since it is the form encountered in any conventional solver: first add up all the components and then apply the preconditioner to the sum. The second one

$$\mathbf{P}^{-1}\mathbf{A}\mathbf{x} \approx \mathbf{P}^{-1}\mathbf{A}_1\mathbf{x} \boxplus \mathbf{P}^{-1}\mathbf{A}_2\mathbf{x} \boxplus \cdots \boxplus \mathbf{P}^{-1}\mathbf{A}_L\mathbf{x}, \tag{7.22}$$

is not encountered in conventional solvers, because it would lead to a huge loss in performance as the preconditioner has to be applied $L$ times now. For a tensor product solver, however, this scheme can be advantageous, which shall be shown in the following.

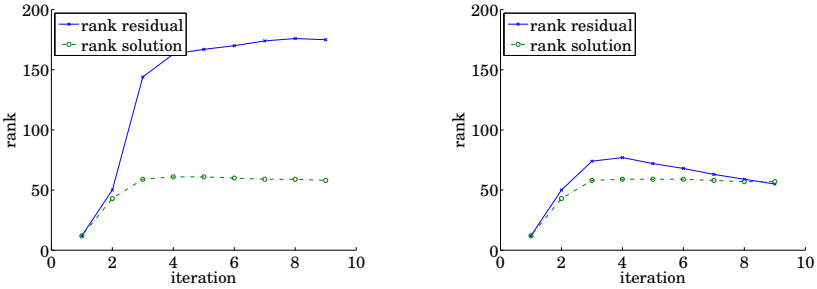



**Figure 7.1:** Numerical rank of the residual and the solution per iteration of the simple iterations solver for the large model with preconditioning strategy basic (left) and with preconditioning strategy inside (right). (Script: `figures/show_ranks_res_and_solution`)

One can observe in Fig. 7.1 that the residual has much higher rank than the solution given the same relative truncation parameter. The reason for this is that the differential operator and its discrete counterpart $\mathbf{A}$ is smoothing, taking a rough right hand side into a much smoother solution. A large fraction in the runtime of a low-rank solver are the orthogonal and the singular value decompositions in the truncation operator, which have a complexity of the tensor rank to the second and third power, respectively. Thus, if the truncations could be performed in the solution space instead of the residual space, it may be possible to reduce that fraction of the runtime considerably.

One way to achieve this is by applying the preconditioner as shown in Eq. (7.22). If the preconditioner $\mathbf{P}$ approximates the

operator **A** well in some sense it should also mirror its smoothing properties. Given the ranks as in Fig. 7.1 the following rough calculation can be made: the ratio of ranks is about three in the mean. As the runtime of the QR decompositions scales with the square of the rank, it can be expected that the speed up of the orthogonalisations will be about nine[3].

| Strategy | basic | dyn/basic | dyn/inside | dyn/ilu |
|---|---|---|---|---|
| Mean rank res. | 506 | 194 | 86 | 69 |
| Mean rank sol. | 110 | 41 | 45 | 40 |
| Final rank res. | 735 | 589 | 200 | 148 |
| Final rank sol. | 127 | 99 | 117 | 98 |
| Iterations | 7 | 10 | 11 | 11 |
| Runtime | 222.1 | 100 | 68.8 | 47.1 |
| Truncations | 193.9 | 77.2 | 41.3 | 31.6 |
| QR | 54.9 | 39.4 | 31.3 | 26.1 |
| SVD | 129.4 | 34 | 7.7 | 4.1 |
| Operator | 8.3 | 4.7 | 14.4 | 4.7 |
| Precond. | 6.7 | 3.7 | 11.4 | 2.7 |
| Memory (MiB) | 154.59 | 127.89 | 60.03 | 46.54 |

**Table 7.2:** Performance comparison of the tensor product solver gsi using different preconditioning strategies. Note that runtime is in seconds and the indentation indicates that the given runtime is a part of the runtime shown further up with lower indentation.

(Script: `solver/cmpsol_large`)

Table 7.2 shows tests that have been performed with this strategy. The first two columns show for comparison memory and runtime statistics for the simple iterations solver (gsi) with fixed and with dynamic truncation (dyn) and standard preconditioner application (strategy basic). The third column shows again simple iterations with dynamic truncation, but this time with the preconditioner applied inside the operator according to Eq. (7.22)

[3]Note, that for small to medium ranks the QR usually dominates the runtime spent in truncations due to the large constant, while SVD becomes only important for larger ranks.

(strategy inside). It can be observed that approx. 30% of the total runtime could be saved using this preconditioning strategy for the given example.

Looking closely at the time measurements shown in it can be seen that some of the time that was saved in the QR and SVD decompositions was spent in extra time in the preconditioner[4]. Considering again the heuristics that led to the introduction of this scheme, it was only important that the preconditioner being pulled inside the operator had similar smoothing properties as the original differential operator. If it is possible to find a preconditioner that has similar smoothing properties but is cheaper to evaluate, larger savings are possible. Let $\mathbf{P}_c$ denote such a preconditioner, than we can rewrite the preconditioner as $\mathbf{P}^{-1} = \mathbf{P}^{-1}\mathbf{P}_c\mathbf{P}_c^{-1}$ and only pull the cheap preconditioner into the operator

$$\mathbf{P}^{-1}\mathbf{A}\mathbf{x} \approx \mathbf{P}^{-1}\mathbf{P}_c(\mathbf{P}_c^{-1}\mathbf{A}_1\mathbf{x} \boxplus \mathbf{P}_c^{-1}\mathbf{A}_2\mathbf{x} \boxplus \cdots \boxplus \mathbf{P}_c^{-1}\mathbf{A}_L\mathbf{x}). \quad (7.23)$$

The remaining issue is to find a good preconditioner $\mathbf{P}_c$ that is on the one hand cheaper to apply (but not necessarily to set up) than $\mathbf{P}$ and on the other still has smoothing properties similar to the expensive one. In this respect no hard and fast rules exist. One possibility that has been explored here is to use incomplete factorisations like the incomplete LU or incomplete Cholesky decompositions.

As not all preconditioners used in this work are symmetric, favour was given to the LU decomposition, especially since the advantages in using Cholesky in this context were minor and could be ignored in terms of total memory consumption and runtime. Further, using the LU decomposition lead to more robust code. The main difference between LU and Cholesky as efficiency is concerned is the setup time and a factor two in storage, which plays only a marginal role in the total resource consumption of the algorithms. A more important fact is, that the time for

[4] Note that in some numerical experiments done by the author the effect was much more pronounced than here.

application is equal, since the same number of back substitutions has to be performed in both cases. Since setup time was a minor factor in the present stochastic problems and due to the greater applicability of the LU factorisation, only the incomplete LU decomposition (ILU) was used. The ILU implementation in Matlab® allowed the following variants:

- `nofill`: the sparsity pattern of the factors $\boldsymbol{L}$ and $\boldsymbol{U}$ is a true subset of the sparsity pattern of the matrix $\boldsymbol{A}$. This variant is also known as ILU(0).
- `droptol`: Instead of dropping all elements in $\boldsymbol{L}$ and $\boldsymbol{U}$ that are zero in $\boldsymbol{A}$, only those are zeroed that are below some parameter $\tau$, the drop tolerance. This leads to better approximations of the inverse than ILU(0).
- `milu`: in the (row-) modified ILU the factors are scaled such that for an all-one vector $\boldsymbol{e}$ the relation $\boldsymbol{A}\boldsymbol{e} = \boldsymbol{L}_{\text{inc}}\boldsymbol{U}_{\text{inc}}\boldsymbol{e}$ holds. This variant is known as MILU and can be combined with the `nofill` and `droptol` options.

**Remark 7.2.** *The best performance in this work could be achieved when setting* `droptol` *to* 0.02 *and using the row-modified ILU. However, the best settings certainly depends on the problem at hand, and manual tuning of the parameters will be needed in each case.*

The preconditioning strategy based on a cheap preconditioner using ILU is referred to as strategy ilu in the rest of this thesis. In the last column of Table 7.2 it can be observed that this strategy still brings some performance gains, as the runtime spent in the preconditioner application drops considerably. For similar results computed for a larger model see also Table 8.4 on page 129.

### 7.2.1 Implementation of preconditioning strategies

For implementing these preconditioner strategies existing code for the solvers need not be changed. Instead, the operator **A** can be modified and then used in the unmodified solver. How the

operator is modified can be seen in Alg. 7.1, setting $\mathbf{P}_s = \mathbf{P}$ for preconditioning strategy inside and $\mathbf{P}_s = \mathbf{P}_c$ for strategy ilu.

**Algorithm 7.1** Preconditioning strategy with preconditioner $\mathbf{P}_s$

**Input:** Operator sum $\mathbf{A} = \sum_{i=1}^{L} \mathbf{A}_i$, preconditioner $\mathbf{P}_s$
**Output:** Modified operator $\tilde{\tilde{\mathbf{A}}}$
1: **for** $i \leftarrow 1, 2, \ldots, L$ **do**
2: $\quad \tilde{\mathbf{A}}_i \leftarrow \mathbf{P}_s^{-1} \circ \mathbf{A}_i$
3: **end for**
4: $\tilde{\tilde{\mathbf{A}}}_i \leftarrow \mathbf{P}_s \circ \left( \sum_{i=1}^{L} \tilde{\mathbf{A}}_i \right)$

Note that forming the matrix product in the algorithm by setting $\tilde{\mathbf{A}}_i = \mathbf{P}_s^{-1}\mathbf{A}_i$ instead of the abstract composition would lead to full matrices and is thus not efficient for large systems. A requirement is hence, that the library used supports abstract composition and inversion of linear operators. In object-oriented frameworks this can easily achieved by implementing a class hierarchy for operators, and composition and inversion as operations thereon return wrapper classes that perform the given operations on the wrapped objects. In sglib this is implemented by cell arrays, representing the linear operators, and the functions `operator_apply`, `operator_compose`, `operator_from_matrix`. The former three are probably self-explanatory; the latter constructs a linear operator that either solves each time it is applied with the Matlab® solve function, or—depending on some Boolean parameter—constructs on initialisation the LU decomposition of the matrix and then uses this for each solve. The full algorithm is implemented in `precondition_system`.

**Remark 7.3.** *One remark has to be made concerning the truncation parameters $\varepsilon_{\mathrm{op}}$ and $\varepsilon_{\mathrm{b}}$. Suppose that $\varepsilon_{\mathrm{op}}$ and $\varepsilon_{\mathrm{b}}$ are chosen to be equal to $\varepsilon_{\mathrm{a}}$. If preconditioning is done only after operator application (i.e. preconditioning strategy basic), truncation in the operator is performed in the space of the residual. Since the operator $\mathbf{A}^{-1}$ is smoothing, the residual is much rougher than the solution, and truncation will thus lead to a significantly larger*

*numerical rank. When truncation is then performed on the solution with parameter $\varepsilon_\mathrm{a}$ the update ratio is still able to detect stagnation, since most of the truncation will occur then.*

*If, however, preconditioning is done inside the operator (like in the preconditioning strategies* inside *and* ilu*), the truncations are done in the same space the solution lives in. In that case the truncation operator $T_\mathrm{a}$ will not reduce the numerical rank significantly anymore, and comes close to the identity. The update ratio then cannot detect stagnation and will falsely indicate that the truncation parameter is sufficiently small for the solver to continue. One remedy for this is to choose $\varepsilon_\mathrm{op}$ and $\varepsilon_\mathrm{b}$ sufficiently smaller than $\varepsilon_\mathrm{a}$, such that enough information on the rougher modes is retained after operator application. The coarsest truncation will then still be performed by $T_\mathrm{a}$ and the update ratio can still detect stagnation. A common choice in this work that worked satisfactorily was to set $\varepsilon_\mathrm{op} = \varepsilon_\mathrm{b} = \varepsilon_\mathrm{a}/10$.*

**Remark 7.4.** *Another strategy can be applied if the spatial preconditioner is based on a multigrid algorithm. If the algorithm does more than one cycle, then some of the cycles could be pulled into the operator, and the remaining cycles be performed after summing up the terms. This has the advantage that the effect of the inner preconditioner does not have to be reversed. Since in this work the spatial dimensions were small enough to do explicit LU decompositions of the spatial preconditioner, this has not been tested experimentally.*

**Remark 7.5.** *The results on perturbed iterations from Chapter 5 can be applied when the estimate for the effective operator truncation in Eq. (6.14) from Section 6.1.2 is adapted for the modified operator used here. A simple estimate shows that*

$$\tilde{\varepsilon}_\mathrm{op} = \varepsilon_\mathrm{op}(1+\varepsilon_\mathrm{op})^{L-1} \,|||\mathbf{P}||| \,(|||\mathbf{P}^{-1}\mathbf{A}_1||| + \cdots + L|||\mathbf{P}^{-1}\mathbf{A}_L|||), \tag{7.24}$$

*which takes on the same form for the two preconditioning strategies* inside *and* ilu*, while the other estimates from Section 6.1.2 stay unchanged.*

# Chapter 8

# Numerical results

*In this chapter the numerical results attained with the methods described in the foregoing chapters on models derived from the stochastic groundwater equation are described. First the different models are presented. Then performance results and comparisons for variations of different parameters that influence convergence and efficiency of the algorithms are presented.*

## 8.1 Models

In order to gauge the performance of the numerical methods developed in this thesis, numerical models of varying size have been used. All of those models rely, however, for ease of presentation on the same continuous model. This reference model is then discretised using different levels of accuracy such that the size of the discrete system ranges from approx. $10^4$ to approx. $10^8$ degrees of freedom (DOF).

### 8.1.1 The continuous model

The partial differential equation employed here is the stationary diffusion equation

$$-\nabla \cdot (\kappa(x,\omega)\nabla u(x,\omega)) = f(x,\omega) \text{ on } \mathcal{D}, \tag{8.1}$$

on the domain $\mathcal{D}$ with Dirichlet and Neumann conditions

$$-u(x,\omega) = g(x,\omega) \text{ on } \Gamma_D \subset \partial\mathcal{D} \tag{8.2}$$

$$-\boldsymbol{n} \cdot (\kappa(x,\omega)\nabla u(x,\omega)) = h(x,\omega) \text{ on } \Gamma_N = \partial\mathcal{D} \setminus \Gamma_D, \tag{8.3}$$

on the boundary $\partial\mathcal{D}$. The domain $\mathcal{D}$ of the problem is the so-called L-shaped domain $\mathcal{D} = [-1,1]^2 \setminus [-1,0]^2$ depicted in Fig. 8.1.

All input random fields are assumed to be spatially homogeneous (in the strict sense) and nonlinear transforms of Gaussian random fields.

- The hydraulic conductivity $\kappa$ has Gaussian covariance function $\mathrm{cov}_\kappa(x,y) = \exp(\|x-y\|^2/l_\kappa^2)$ with $l_\kappa = 0.8$. The marginal density of $\kappa$ is a Beta distribution with parameters $a = 2.52$ and $b = 0.39$ shifted by $s_\kappa = 0.001$, i.e. $\kappa(x,\cdot) - s_\kappa \sim \mathrm{Beta}(a,b)$ for all $x \in \mathcal{D}$. The coefficient of variation for those parameters is 0.2. A sample realisation of this field is shown in Fig. 8.2(a).

- The right hand side $f$ has exponential covariance function $\mathrm{cov}_f(x,y) = \exp(\|x-y\|/l_f)$ with $l_f = 1.8$. The marginal density of $f$ is a uniform distribution between -1 and 1, i.e. $f(x,\cdot) \sim U(-1,1)$ for all $x \in \mathcal{D}$. A sample realisation of this field is shown in Fig. 8.2(b).

- The Dirichlet boundary conditions were chosen as $g(x,\omega) = x_1(1-x_2^2)$, where the Dirichlet boundary was $\Gamma_D = \partial\mathcal{D} \cap \{(x_1,x_2) \in \mathbb{R}^2 | x_1 > -1\}$. On the Neumann boundary $\Gamma_N = \partial\mathcal{D} \cap \{(x_1,x_2) \in \mathbb{R}^2 | x_1 \leq -1\}$ homogeneous boundary conditions were chosen, i.e. $h(x,\omega) = 0$. The boundary conditions are depicted in Fig. 8.1

### 8.1.2 Discretisation parameters

For the discretisation of the continuous model different levels of accuracy have been defined, leading to discrete models of varying sizes. The differently sized models are used for the analysis of the numerical algorithms depending on the aspect under investigation.

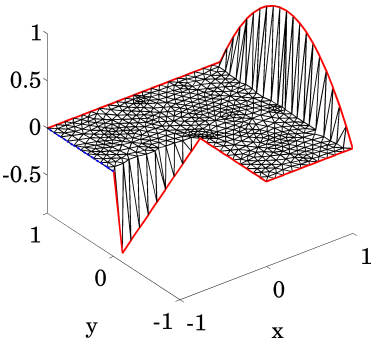
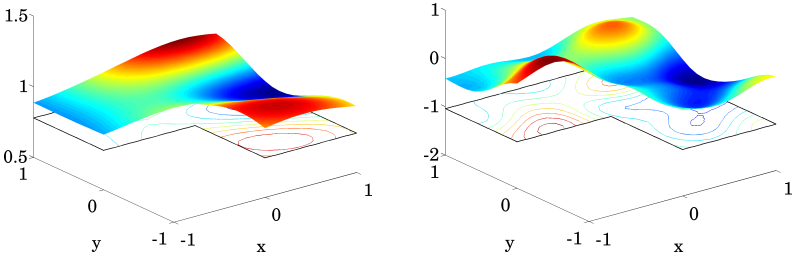



**Figure 8.1:** Boundary conditions for the model problem. The Dirichlet boundary $\Gamma_D$ is indicated by the solid, red line, the Neumann boundary $\Gamma_N$ by the broken, blue line.

(Script: `figures/show_model_bcs_and_solution`)

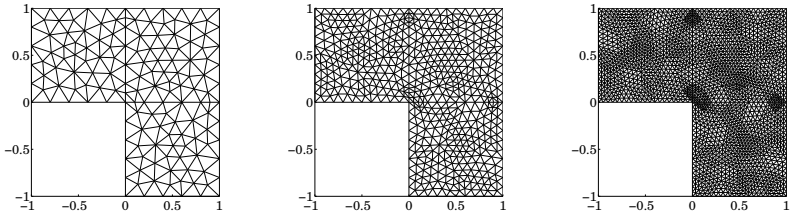



(a) Diffusivity $\kappa$ (b) Right hand side $f$

**Figure 8.2:** Sample realisations of the input random fields $\kappa$ (left) and $f$ (right). (Script: `ranfield/show_input_random_fields`)

For example, some studies need relatively small models for making the quantities of interest computationally feasible, while others try to reach the limits of what is possible with the algorithms developed. For studies of parameter dependence these models will serve as a basis with usually only one or two parameters simultaneously modified.

For the spatial discretisation the mesh in Fig. 8.3(a) was used with different levels of global refinement as depicted in Fig. 8.3(b) and Fig. 8.3(c). For the huge model an even finer mesh has been used, which is not depicted here. However, for the discretisation of the input random fields in this model still the mesh in Fig. 8.3(c) was used and the data then interpolated onto the finer mesh (as no optimised algorithm for computing the KLE has been used, and the smoothness of the fields allowed for this approach).



(a) Coarse mesh ($N = 110$)
(b) Medium mesh ($N = 477$)
(c) Fine mesh ($N = 2145$)

**Figure 8.3:** The L-shaped domain with three different levels of discretisation. $N$ denotes the number of inner nodes.

(Script: `figures/show_geometry`)

The parameters used for the stochastic discretisation of the input random fields are summarised in Tab. 8.1. Those for the solution and the size of the resulting discrete system are summarised in Tab. 8.2. The intended uses of the full discretisation models (i.e. spatial plus stochastic discretisation parameters) are described below.

- The *small model* is used for verification of certain aspects of the algorithms where the full system matrix $\mathbf{K}$ can be

computed explicitly and computationally expensive operations on $\mathbf{K}$ can be performed. For example, to be able to compute all of the quantities for the different stochastic preconditioners in Table 7.1 the small model was needed, as the computation of the spectral radius of the preconditioned operator becomes quickly infeasible for larger models.

- The *medium model* is also used for verification of certain aspects of the algorithms, in which the full system matrix needs to be computed, but the complexity of the algorithms is not as high as of those, for which the small model is needed. Further, it is used for most of the plots depicting the random fields graphically like Fig. 8.2 or Fig. 8.4, since the accuracy of this model is sufficient for these purely demonstrative purposes.

- The *large model* is intended for comparisons with standard solvers (like PCG acting on a full tensor product representation), where the standard solver is at or near its limit due to memory limitations.

- The *huge model* is used to demonstrate the solution of systems using the low-rank tensor methods developed in this thesis, which are not feasible to solve on standard PC hardware using conventional methods (see also Remark 8.1).

**Remark 8.1.** *Note that it would of course be possible to solve also the huge model with standard solvers on larger hardware with larger or distributed memory, or by employing swapping to hard disk to virtually enlarge memory capacity. However, whatever capable the hardware is, it will always be interesting to solve "larger" problems. Thus, no attempt has been made to apply or port the software developed to high performance computing machinery. Without much doubt the methods developed here can also be used to increase the speed and extend the limitations of solving stochastic problems on larger machines as they can on standard PC hardware.*

For computation of errors these systems (except, of course, the huge model) have been solved with a standard PCG solver up

| Model | $l_\kappa$ | $m_\kappa$ | $p_\kappa$ | $\%\text{var}_\kappa$ | $l_f$ | $m_f$ | $p_f$ | $\%\text{var}_f$ |
|---|---|---|---|---|---|---|---|---|
| small | 3 | 3 | 2 | 71 | 4 | 4 | 2 | 81.7 |
| medium | 5 | 5 | 3 | 85.2 | 6 | 6 | 3 | 86.0 |
| large | 10 | 10 | 3 | 97.3 | 20 | 20 | 3 | 93.1 |
| huge | 15 | 15 | 3 | 99.2 | 30 | 30 | 3 | 94.4 |

**Table 8.1:** Parameters for the stochastic discretisation of the input random fields $\kappa$ and $f$ for the the numerical models. $l$ denotes the number of terms retained from the KLE, and $m$ and $p$ are the number of Gaussians and the polynomial degree for the polynomial chaos expansion for the respective field. %var is the percentage of the variance of the field captured with those parameters. (Script: `figures/show_table_models`)

| Model | $m_u$ | $p_u$ | $M$ | $N$ | $MN$ |
|---|---|---|---|---|---|
| small | 7 | 2 | 36 | 124 | 4,464 |
| medium | 11 | 3 | 364 | 506 | 184,184 |
| large | 30 | 3 | 5,456 | 2,044 | 11,152,064 |
| huge | 45 | 3 | 17,296 | 8,216 | 142,103,936 |

**Table 8.2:** Parameters for the discretisation of the solution and resulting degrees of freedom. The number of Gaussians is the sum of those for the fields $\kappa$ and $f$, i.e. $m_u = m_\kappa + m_f$. $M = (m_u + p_u)!\,/(m_u!\,p_u!\,)$ denotes the stochastic, $N$ the spatial, and $MN$ the total number of DOFs.

(Script: `figures/show_table_models`)

to a relatively fine tolerance of $10^{-12}$ for the residual. The mean and standard deviation of a reference solution computed with the medium model is shown in Fig. 8.4.

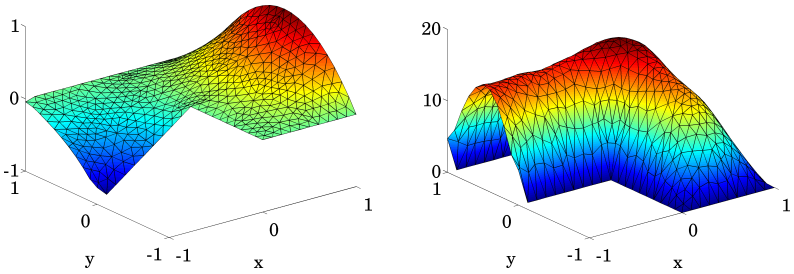



**Figure 8.4:** Mean (left) and standard deviation (right) of the stochastic Galerkin solution for the large model computed with a standard PCG solver and threshold for the relative residual $10^{-12}$. (Script: `figures/show_model_bcs_and_solution`)

## 8.2 Solvers and solver parameters

Linear solvers were implemented as described in Section 6.1.1 and Section 6.2.2 for the simple iterations and the preconditioned conjugate gradients[1]. Since the implemented solvers work on arbitrary vectors, which can be vectors in the classical sense or have tensor product structure (i.e. they just need to be vectors in the mathematical sense of a vector space and the operations defined thereupon, not in the sense of an $n$-tuple of real numbers), they are called here *generalised solvers*.

To easily refer to them in the following, some abbreviations are introduced. The solver using simple iterations is called gsi for *generalised simple iterations*, the solver using preconditioned conjugate gradient iterations is called gpcg. When these solvers are compared to the traditional PCG method this is simply called pcg

[1] Reference implementations can be found in sglib in the files `solver/generalised_solve_simple` and `solver/generalised_solve_pcg`.

The standard preconditioner generally used here is the mean based preconditioner described in Section 7.1.1. If different preconditioners are compared they are indicated by mean for the mean based, kron for the Kronecker product (see Section 7.1.2), and ikron for the inverse Kronecker product preconditioner (see Section 7.1.3).

If not otherwise noted, none of the preconditioning strategies mentioned in Section 7.2 are used. If the strategies are compared, basic stands for using the preconditioner outside the operator, inside for using the standard preconditioner inside the operator, and ilu for using an ILU-based preconditioner inside the operator. The standard options for the ILU-preconditioner are, unless otherwise noted, to use `type='ilutp', droptol=2e-2, milu='row', udiag=1`. For the exact meaning of the parameters see the Matlab® documentation [60].

The abbreviations for the solvers and the solver options and strategies are summarised in Tab. 8.3.

| | |
|---|---|
| gsi | generalised simple iteration solver |
| gpcg | generalised preconditioned conjugate gradients |
| pcg | standard preconditioned conjugate gradients |
| mean | mean based preconditioner |
| kron | kronecker product preconditioner |
| ikron | inverse kronecker product prec. |
| basic | preconditioner is applied normally |
| inside | preconditioner applied inside operator |
| ilu | ILU preconditioner inside operator |

**Table 8.3:** Abbreviations used to describe the solvers, solve options and preconditioning strategies.

## 8.3 Convergence

In this section convergence of the solvers gsi and gpcg is examined. Comparisons are done for different truncation parameters, and the usefulness of the error estimators is evaluated.

Figure 8.5 shows the relative residual $\|r^{(k)}\|/\|r^{(0)}\|$ and the relative error $\|e^{(k)}\|/\|x\|$ as a function of the iteration number $k$. The plots are made for different values of the truncation parameter $\varepsilon$ from $10^{-5}$ to $10^{-2}$ in logarithmic steps of size $\sqrt{10}$. It can be seen clearly how the solvers go into stagnation depending on the size of the truncation parameter. Further, the figures show that the residual stagnates earlier than the error and is thus not a good indicator for stagnation. Figure 8.6 shows the same for the generalised PCG. Convergence is quicker compared to the simple iterations, but shows some oscillations when stagnation is reached. The reason for these oscillations is not yet clear to the author, but does not seem to influence overall convergence.

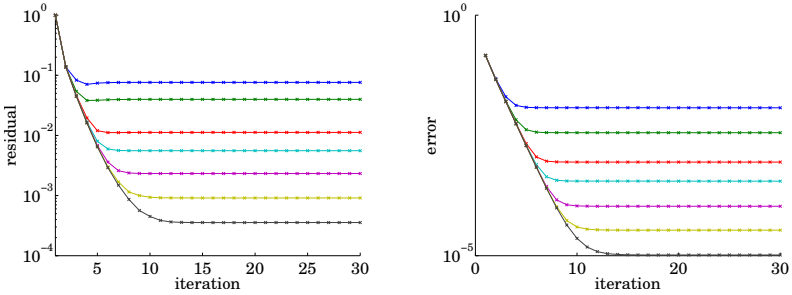



**Figure 8.5:** Relative residual (left) and relative error (right) per iteration for the generalised simple iterations (gsi) for truncation parameters $\varepsilon$ from $10^{-2}$ to $10^{-5}$ (from top to bottom).

(Script: `figures/show_residual_and_error_gsi`)

Fig. 8.7(a) shows the relative residual, relative error, a posteriori error estimate and the update ratio for a truncation parameter $\varepsilon = 10^{-4}$ over 20 iterations. The figure shows again that the relative residual stagnates earlier then the relative error. Furthermore, it can be seen that the relative error and the a posteriori error estimate match well and that the deviation of the update ratio from one is also closely correlated with the stagnation of the error. Fig. 8.7(b) shows the same for the generalised PCG.

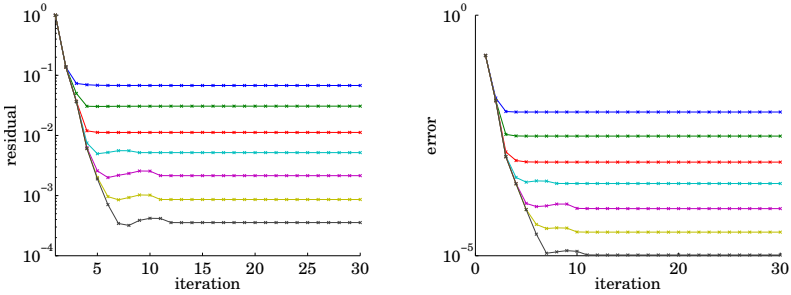
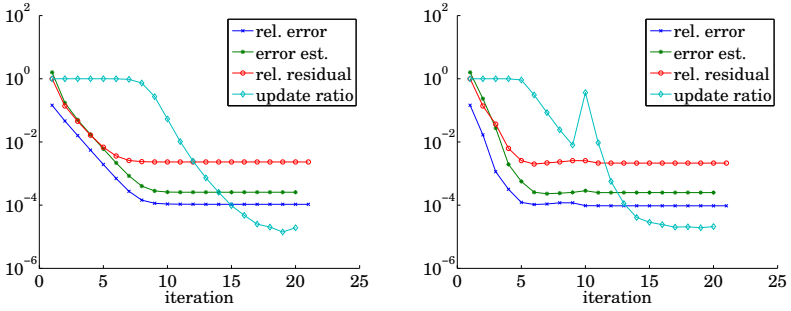



**Figure 8.6:** Relative residual (left) and relative error (right) per iteration for the generalised PCG solver (gpcg)for truncation parameters $\varepsilon$ from $10^{-2}$ to $10^{-5}$ (from top to bottom).

(Script: `figures/show_residual_and_error_gpcg`)




(a) Solver gsi (b) Solver gpcg

**Figure 8.7:** Relative residual, relative error, a posteriori error estimate and update ratio for the generalised simple iterations (left) and for PCG (right) per iteration for the truncation parameters $\varepsilon = 10^{-4}$.

(Script: `figures/show_update_ratio_and_posterior_err`)

### 8.3.1 Comparison of KLE modes

The KLE modes computed in a tensor product solver are supposed to match those computed a posteriori from a full PCE solution. The result of such a comparison is shown in Fig. 8.8.

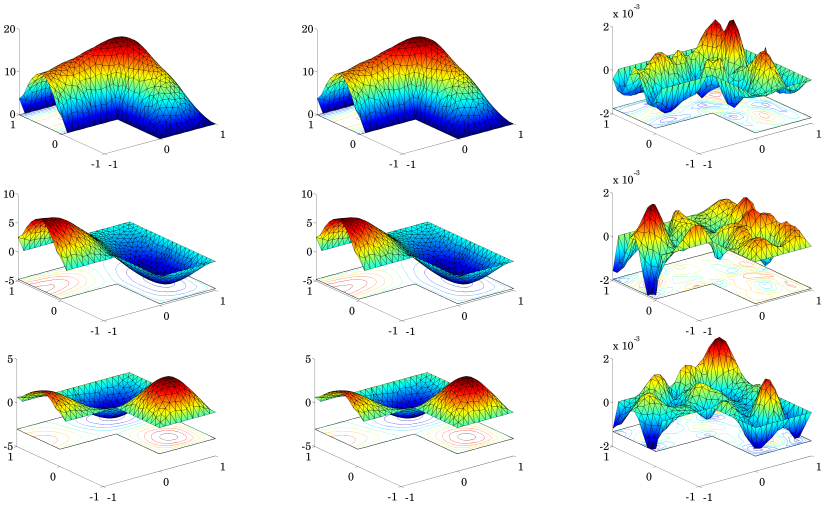


**Figure 8.8:** Comparison of KLE modes between the normal solver and the low-rank solver. On the left are the first three modes computed from a full solution with a standard PCG solver with tolerance set to $10^{-6}$; in the middle are the corresponding modes directly taken from the generalised simple iteration solver with dynamic truncation and tolerance set to $10^{-2}$; on the right the absolute difference between the modes is shown.

(Script: `figures/show_compare_modes`)

The KLE modes on the left are computed from a full PCE solution with a standard PCG solver with tolerance set to $10^{-6}$. From this solution the first three KLE modes were computed using the algorithm described in Section 3.3.2. In the middle column the corresponding modes were directly taken from the simple iteration solver using the canonical format with dynamic truncation (minimal truncation parameter $\epsilon_{\min} = 10^{-8}$) and tolerance set to $10^{-2}$. The modes were scaled such that both

had the same $L_2$-norm and multiplied by $-1$ if the inner product between corresponding modes was negative, in order to make the modes visually comparable without changing their shape. Comparison between the left and middle column show almost no difference as can be seen in Fig. 8.8. In the right column the difference between the modes was computed and shows that the difference is about 2 to 3 orders of magnitude smaller than the modes themselves. In other words, the solution computed with the low-rank solver is indeed a good approximation of the KLE of the full solution.

## 8.4 Large systems

How the tensor product solvers performs on large systems was tested with the huge model of Section 8.1.2, containing approx. $1.4 \cdot 10^8$ degrees of freedom. Since this model could not be solved with standard solvers, only results with the tensor product solver are supplied.

Table 8.4 shows in the left columns a comparison between the solver gsi using the standard QR algorithm in the truncation algorithm and using the optimised one described in Section 4.1.5. It can be observed that the runtimes cost of the QR decompositions could be reduced by approx. 60% leading to a total runtime reduction of about 15%. Note that this effect could only be observed for really large models like this one.

Table 8.4 further shows in the rightmost columns that the rank of the residual becomes much smaller when one of the preconditioning strategies described in Section 7.2 is applied (columns "dyn/inside" and "dyn/ilu"). It can also be seen that when the normal preconditioner is used inside the operator the runtime for preconditioner applications grows severely (column "dyn/inside"), but is largely reduced when the cheap inner preconditioner (column "dyn/ilu") is used. This leads to a reduction in runtime by a factor of approx. 2.5 and a reduction in memory of more than 10, showing convincingly the potential of the method.

It can further be observed that dynamic truncation leads to a

| Mode/strategy | fix qr | fix optqr | dyn/basic | dyn/inside | dyn/ilu |
|---|---|---|---|---|---|
| Mean rank res. | 963 | 963 | 332 | 110 | 99 |
| Mean rank sol. | 135 | 135 | 56 | 55 | 56 |
| Final rank res. | 1364 | 1364 | 1085 | 254 | 208 |
| Final rank sol. | 150 | 150 | 140 | 139 | 138 |
| Iterations | 8 | 8 | 11 | 11 | 12 |
| Runtime | 3766 | 3183 | 1571 | 882 | 601 |
| Truncations | 3255 | 2689 | 1277 | 514 | 480 |
| QR | 1501 | 940 | 775 | 442 | 421 |
| SVD | 1436 | 1427 | 379 | 32 | 23 |
| Operator | 214 | 213 | 105 | 297 | 51 |
| Precond. | 199 | 199 | 96 | 274 | 39 |
| Memory (MiB) | 894 | 882 | 729 | 93 | 60 |

**Table 8.4:** Performance comparison of the solver gsi with fixed truncation using standard qr and optimised qr decomposition in the left two columns, and with dynamic truncation using different preconditioning strategies in the three rightmost columns. Note that runtime measurements are in seconds and the indentation indicates that the given runtime is a part of the runtime shown further up with lower indentation. (Script: `solver/cmpsol_huge`)

slight increase in the number of iterations compared to truncations with a fixed truncation parameter. However, the total runtime is still much lower for the former, since the cost per truncation and thus also per iteration is much smaller. Possibly the number of iterations could be reduced by choosing a smaller initial truncation parameter for the dynamic truncation, or by reducing it more aggressively; however, it is questionable whether this will reduce runtime further, as most of the additional iterations are performed with relatively low rank, and thus very fast.

## 8.5 Performance

In this section the performance of the low-rank solvers is compared to that of a conventional PCG solver. The discrete systems used

are relatively large, but only such that low-rank and conventional methods both work. The relative performance of the solvers depends on a number of parameters, of which the most important are:

- Threshold of the relative error: While both kinds of solvers need to iterate longer to achieve a small error, low-rank solvers can use approximations of even lower rank if the target error is not tight. Hence, it can be expected that full-rank solvers may perform better on tight error goals, while low-rank solvers have an advantage for looser error goals.

- Number of KLE modes in the operator: The effect of this parameter should be small, as the runtime of both types of solvers depends linearly on the number of modes.

- Number of spatial and stochastic degrees of freedom: From the asymptotic behaviour of the different solver types, as outlined in Section 4.3, the relative performance advantage of low-rank solvers should increase with system size. On the opposite, for smaller system sizes conventional solvers should have the advantage.

In the following, after a brief discussion on how the runtime and memory measurements were performed, it will be shown how variation of these parameters affects relative performance of the solvers. The base model and discretisation parameters chosen were those of the large model (see Section 8.1.2).

### 8.5.1 Runtime and memory measurements

In order to make the measurements in this work comparable, all timings were performed on the same hardware. The computer used had a 64bit PC architecture with 2 Intel® Xeon® 5160 CPUs with 4 cores running at 3.00GHz and 4 MiB L2 cache. The total memory of the machine was 8 GiB of RAM. The operating system used was GNU/Linux 2.6.31.6.

The runtime measurements were performed by using the Matlab® `tic` and `toc` functions. `tic` and `toc` measure the so-called wall clock time, i.e. the time that could be measured by someone doing the measurements manually with a stopwatch. This time measurement then includes also CPU time spent by other processes. However, time saved by Matlab® by automatically distributing work over several cores on multi-core CPUs is not counted herein. This is in contrast to the `cputime` function, which measures only the CPU time used by the Matlab® process, but sums up the times used on all cores. This measure can thus also be higher than the pure wall clock time (see Example 8.2).

**Example 8.2.** *For inverting a* $4000 \times 4000$ *matrix* `tic/toc` *reports approx.* 7 s*, while* `cputime` *reports approx.* 26 s*. When two parallel Matlab® processes perform this computation at the same time, the reported times are* 15 s *and* 32 s*, showing that the wall clock time doubles, as expected, while the CPU time increases much less significantly. The increase therein can probably be attributed to more cache misses or other memory related issues.*

Which kind of measurement to use is a subjective issue, since arguments can be made for both. The argument for using CPU time measurements is that all operations are taken into account as if the code was executed on a single-threaded CPU, making for a somewhat more "objective" measurement. The argument for wall clock timings is that this is what really matters to the user, giving advantage to better parallelised code. As the second argument was considered more important by the author, in all performance tests wall clock time was used as the measure.

Memory was measured using process information via the proc file system (procfs). The file `/proc/<pid>/status`, where `<pid>` is the process id of the main Matlab® process, was read and the field `VmSize` parsed. This was done during the whole solve process, and the difference between the initial `VmSize` and the maximum `VmSize` was recorded. Note that memory measurements on the 64bit systems turned out to be unreliable and had thus been performed on 32bit workstations that returned more consistent results.

### 8.5.2 Dependence on relative error threshold

The dependence of solve time on the target accuracy of the numerical solution depends on the type of solver and is in general not linear for all kinds of solvers. Thus, showing only the runtimes for different solvers with some arbitrary fixed error threshold does not allow for a fair comparison of those solvers. Raising or reducing the threshold will not decrease or increase the runtime of different solvers by the same factor.

Just to give a few examples: since solvers using simple iterations usually converge linearly, the runtime is roughly proportional to the logarithm of the threshold. Krylov subspace methods like conjugate gradients show often super-linear convergence thus comparing most favourably for very low thresholds. For the type of solvers shown here, not only the number of iterations depends on the threshold as for the standard methods, but also the work per iteration, since smaller truncation parameters can be used. Thus these methods are expected to perform comparably best for relatively low demands on accuracy.

| res. ($\times 10^{-4}$) | 1.00 | 3.00 | 10.00 | 30.00 | 100.00 |
|---|---|---|---|---|---|
| gsi | 1518.06 | 677.41 | 220.51 | 96.64 | 41.21 |
| gsi dyn | 506.42 | 336.37 | 97.79 | 64.53 | 35.13 |
| gsi dyn/ilu | 161.22 | 76.91 | 46.32 | 38.35 | 21.43 |
| gpcg | 5352.35 | 2988.25 | 1262.70 | 411.55 | 79.33 |
| gpcg dyn | 4115.71 | 2876.67 | 979.43 | 459.15 | 78.88 |
| gpcg dyn/ilu | 1266.09 | 467.29 | 282.66 | 111.31 | 56.36 |
| pcg | 128.43 | 112.64 | 88.77 | 74.21 | 54.92 |

**Table 8.5:** Comparison of the runtime of the generalised simple iteration solver with the standard PCG solver for different values of the threshold for the relative residual.

(Script: `solver/cmpsol_large_rel_res`)

Table. 8.5 shows the runtime for the different solvers for different thresholds on the relative residual. It can be seen that the runtime for the standard PCG solver is better than for most of the low-rank solvers except for the gsi dyn/ilu case, in which it is only fast

for relative residuals of about $10^{-4}$ and smaller. However, for all low-rank solvers it is observable that the runtime decreases much faster for higher thresholds.

**Remark 8.3.** *The table further shows that the low-rank PCG solver* gpcg *is many times slower than the solver* gsi*. The reason for this is not yet clear to the author and subject of further investigation.*

Table 8.6 shows the relative residual actually attained by the solvers for comparison. The residuals attained are all below the prescribed threshold as it should be, but not significantly below, so that no performance difference between the solvers can be attributed to this.

| res. ($\times 10^{-4}$) | 1.00 | 3.00 | 10.00 | 30.00 | 100.00 |
|---|---|---|---|---|---|
| gsi | 0.90 | 2.72 | 9.58 | 17.83 | 75.63 |
| gsi dyn | 0.98 | 1.74 | 9.22 | 16.66 | 76.16 |
| gsi dyn/ilu | 0.64 | 2.39 | 9.61 | 16.99 | 66.54 |
| gpcg | 0.71 | 2.59 | 8.81 | 18.70 | 53.99 |
| gpcg dyn | 0.90 | 1.60 | 8.07 | 15.03 | 76.87 |
| gpcg dyn/ilu | 0.58 | 2.95 | 5.18 | 26.74 | 55.95 |
| pcg | 0.40 | 2.11 | 8.07 | 16.95 | 49.56 |

**Table 8.6:** Relative residual attained for the generalised simple iteration solver and the standard PCG solver for different values of the threshold for the relative residual.

(Script: `solver/cmpsol_large_rel_res`)

### 8.5.3 Dependence on number of operator terms

The runtime of both standard and tensor methods depends on the number $l_k$ of terms in the stochastic operator.

Tab. 8.7 shows the dependence of the solver performance for different numbers of terms in the operator. The runtime for both types of solvers seems to grow linearly plus some constant, where the constant term for the PCG solver is much higher, however.

| $l_k$ | 2 | 3 | 4 | 6 | 8 | 10 | 12 | 14 | 17 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|
| gsi | 7 | 12 | 23 | 49 | 116 | 188 | 268 | 349 | 482 | 736 |
| gsi dyn | 10 | 18 | 23 | 44 | 67 | 136 | 211 | 269 | 360 | 453 |
| gsi dyn ilu | 10 | 14 | 17 | 29 | 37 | 58 | 69 | 80 | 125 | 160 |
| gpcg | 13 | 24 | 46 | 141 | 599 | 1084 | 1631 | 1971 | 2598 | 3153 |
| gpcg dyn | 34 | 43 | 102 | 298 | 809 | 1446 | 2022 | 2388 | 3473 | 6062 |
| gpcg dyn/ilu | 36 | 38 | 51 | 111 | 162 | 394 | 566 | 598 | 1014 | 1583 |
| pcg | 50 | 50 | 53 | 60 | 81 | 87 | 95 | 101 | 112 | 118 |

**Table 8.7:** Comparison of the runtime of the generalised simple iteration solver with the standard PCG solver for different numbers of terms in the linear operator ($l_k$).

(Script: `solver/cmpsol_large_op_klterms`)

This is probably due to the preconditioner application, which does not depend on the number of terms in the operator. Otherwise, runtimes grow for both at a slightly superlinear rate. That the rate is not exactly linear is probably due to the fact that the operator itself becomes more difficult to solve with more terms and thus more iterations are needed.

### 8.5.4 Dependence on system size

For standard solvers the runtime depends often almost linearly on the system size (given a fixed band-width of the matrix), since all time consuming operations like matrix-vector products, inner products, vectors additions and multiplication by scalars depend linearly on the size of the vectors[2]. This is not so for the tensor methods discussed here, since for increasing system size (i.e. the total number of degrees of freedom $NM$) the rank $K$ of the tensor quantities does not usually increase and thus the effective system size for these solvers ($K(N+M)$) does not increase linearly. Thus for larger system sizes tensor product methods should have a performance advantage over standard methods, while for small

[2] Assuming that the condition number does not change much with the system size.

systems the standard methods should have an advantage.

| $M$ | 50 | 200 | 450 | 800 | 1250 | 1800 | 2450 | 3200 | 4050 | 5000 |
|---|---|---|---|---|---|---|---|---|---|---|
| gsi | 6 | 21 | 50 | 109 | 170 | 238 | 264 | 306 | 315 | 350 |
| gsi dyn | 2 | 21 | 38 | 58 | 71 | 78 | 102 | 100 | 158 | 176 |
| gsi dyn ilu | 9 | 19 | 27 | 34 | 57 | 54 | 63 | 72 | 76 | 81 |
| gpcg | 8 | 40 | 95 | 294 | 646 | 875 | 1141 | 1413 | 1576 | 1761 |
| gpcg dyn | 6 | 53 | 105 | 297 | 459 | 567 | 636 | 1077 | 1251 | 1352 |
| gpcg dyn/ilu | 16 | 54 | 63 | 118 | 183 | 303 | 337 | 390 | 410 | 444 |
| pcg | 1 | 4 | 7 | 18 | 27 | 39 | 52 | 63 | 79 | 94 |

**Table 8.8:** Comparison of the runtime of the generalised simple iteration solver gsi with the standard PCG solver for different values of the stochastic dimension $M$ between 50 and 5000 corresponding to total system size between approx. $2.7 \cdot 10^5$ and $2.7 \cdot 10^7$. (Script: `solver/cmpsol_large_system_size`)

Table 8.8 shows very well the expected behaviour for the standard PCG solver, where the runtime scales nearly exactly linearly with $M$. For the low-rank solver only the solver gsi dyn/ilu shows plainly the sub-linear behaviour, and to some part also gpcg dyn/ilu. The "advantage" alluded to earlier cannot be seen in the absolute numbers, but rather in the relative ones. The ratio in runtime between the smallest and the largest system is about 90 for PCG and about 9 for the solver gsi dyn ilu.

## 8.6 Summary

In summary the low-rank solver gsi showed good performance, especially when combined with dynamic truncation and preconditioning strategy ilu. Even huge systems with more than $10^8$ degrees of freedom could be solved efficiently. However, this only holds if the termination criterion is not too stringent as otherwise rank growth will severely affect the runtime.

The low-rank PCG solver gpcg did not show equally satisfactory performance. It was between 2 and 15 times slower than the simple iteration solver. The reason could lie in excessive rank growth

in the preconditioned residual $\boldsymbol{z}^{(k)}$; however, this needs further research. Furthermore, `gpcg` often showed severely degraded convergence when combined with dynamic truncation.

The effect of the optimised truncation algorithm could be seen, but played only a minor role. Here, further optimisation could be possible by directly invoking the LAPACK/BLAS interfaces from Matlab®.

# Chapter 9

# Conclusions and outlook

*This chapter summarises the main goals that have been achieved in this thesis. Further, a brief outlook on questions that have emerged during this work or on alternatives and extensions to the methods presented is given. The chapter concludes with a short personal evaluation of the methods.*

## 9.1 Conclusions

The main goals that have been achieved during the work on this thesis:

- For the sequences of iterates in iterative processes that are subjected to frequent perturbations—which is the case when tensor formats are employed—stagnation of the perturbed sequences in a neighbourhood of the solution and error estimates could be deduced and proved. A criterion for detection whether the solver enters stagnation has been developed. Moreover, the practical applicability of the results has been demonstrated.

- Estimates for the combined truncations that occur during single iteration steps of the simple iterations solver could be derived, so that the convergence results for the perturbed iterations could be applied. For conjugate gradients this could also be achieved, however, but limited to the convergence estimate that is equivalent to that of the steepest descent method.

- Tensor formats and the corresponding arithmetic and truncation methods have been implemented and integrated into linear solvers for stochastic PDEs. This has been done for simple iterations and for the conjugate gradient method. The solvers were used to solve systems more than $10^8$ degrees of freedom efficiently. Furthermore, for medium sized methods a performance comparable to that of the conjugate gradient method on standard representations could be attained.

- Different strategies for the tensor truncations have been analysed and tested in numerical experiments. It could be demonstrated that dynamic truncation strategies lead to considerable performance improvements in the solvers. Optimisations in the truncation algorithm have also been analysed and shown to provide some performance gains for large systems.

- Different preconditioning strategies have been implemented and tested, showing that application of the preconditioner—or a similar but less expensive approximation of it—inside the operator can lead to noticeable performance improvements of the tensor solver due to reduced numerical rank of the tensor iterates.

- A complete framework for the discretisation and synthesis of random fields, post-processing (plotting, probability densities, moment computations), computation and solution of the discretised equations etc. for stochastic Galerkin methods has been developed (sglib). The methods therein work on standard representations as well as on tensor representations like the canonical format. The framework has been highly optimised to allow the computation of relatively large systems, and has been supplied with a unit testing framework and an extensive set of unit tests to enhance the reliability of the computations.

Some minor results that should also be mentioned here:

- The connection between the Karhunen-Loève expansion and the matrix SVD of the coefficient matrix of discretised random fields has been analysed in detail to make it usable in numerical algorithms. Based on this, algorithms have been devised and implemented that compute the KLE via the generalised SVD for discretised random fields in different formats.
- A new algorithm to compute inner products between tensors or the norms of tensors which is also accurate if the result is close to zero has been developed and implemented.
- A new preconditioner has been developed that has better characteristics for simple iterative methods than the better known mean-based or Kronecker preconditioners. However, this preconditioner is much too expensive to set up, and is thus not currently competitive with those. Maybe this can be fixed in some future work.

## 9.2 Outlook

During the course of writing this thesis and implementing the relevant software some problems and questions emerged that need further investigation:

- The performance of the tensor PCG was often much worse than for the simple iterations. Further, convergence frequently broke down completely when dynamic truncation was employed. Deeper analysis of this behaviour is necessary and may lead to methods that can recover the good convergence properties of the conventional PCG.
- As there is much research going on in the field of higher order tensor decompositions and algorithms, it needs to be constantly reevaluated how these perform when employed in the iterative methods developed in this work.
- In the context of stochastic PDEs many methods to cope with the high dimensionality of the discrete problems are

currently investigated, such as model reduction methods, rank-1 updates, spectral decompositions or tensor based methods like the one developed in this thesis. However, due to the complexities involved in implementing each of these methods, no detailed comparisons have yet been made. In order to facilitate the comparison of those methods frameworks have to be developed which allow more abstract, high-level views on those methods while still being efficient and thus making it easier to compare different approaches in terms of efficiency, accuracy of approximation, and possible limitations.

- During the course of this thesis it became apparent that methods for the synthesis of random fields according to prescribed stochastic properties are still not sufficiently reliable. In the author's view it would be a worthwhile endeavour to develop methods that can automatically adapt discretisation parameters for random fields based on the stochastic parameters of the field and given accuracy requirements on the approximation.

### 9.2.1 Evaluation

The following is the author's personal evaluation and opinion concerning the methods discussed in this thesis, resulting from personal experience with those methods.

Tensor based methods show great potential to compute large stochastic problems given that the smoothness of the solution allows approximations with small rank and that the residuals during the computation also only have small rank. This is in general only given, if the solution is only needed with low to medium accuracy, and when the stochastic variability is not too high. The latter point is usually determined by the coefficient of variation of the conductivity field and by the type of boundary conditions. For example, deterministic Dirichlet boundary conditions force the variance on the boundary to zero, while Neumann boundary

conditions do not, which leads to much higher variability in the solution and the residuals.

However, for problems with high stochastic variability the performance of the methods tends to deteriorate severely. This is caused by the scaling behaviour of the truncation algorithm, which has at least quadratic complexity in the numerical rank. Due to this sensitivity to the numerical rank the robustness of the methods is in general not as high as would be desirable. All tensor methods developed thus need close monitoring concerning the growth of the numerical rank and actual convergence behaviour, and often need much tweaking of parameters to achieve convergence and sufficient performance. Especially if the preconditioning strategies for lowering the residual rank discussed in Section 7.2 are used, convergence has to be checked for false positives, as the residual norm is then often underestimated. As an overall conclusion it can be said that low-rank methods show great potential for solving large stochastic problems, although much work on robustness of those methods is still needed.

# Appendix A

# Notation

## A.1 Symbols

### Matrices and Vectors

| | |
|---|---|
| $\boldsymbol{x}$ | a vector |
| $[\boldsymbol{x}]_i$ | entry of vector $\boldsymbol{x}$ in row $i$ |
| $\boldsymbol{A}$ | a matrix |
| $[\boldsymbol{A}]_{i,j}$ | entry of matrix $\boldsymbol{A}$ in row $i$ and column $j$ |
| $\boldsymbol{A}^T$ | transpose of matrix $\boldsymbol{A}$, i.e. $[\boldsymbol{A}^T]_{i,j} = [\boldsymbol{A}]_{j,i}$ |
| $\boldsymbol{I}$ | the identity matrix $[\boldsymbol{I}]_{i,j} = \delta_{i,j}$ |
| $\lambda_i$ | $i$-th eigenvalue of matrix $\boldsymbol{A}$ |
| $\sigma_i$ | $i$-th singular value of matrix $\boldsymbol{A}$, i.e. square roots of the eigenvalues of $\boldsymbol{A}^T\boldsymbol{A}$ |
| $\boldsymbol{\sigma}$ | sequence of singular values |
| $\rho(\boldsymbol{A})$ | spectral radius of matrix $\boldsymbol{A}$, i.e. $\rho(\boldsymbol{A}) = \max_i(\lvert\lambda_i\rvert)$ |
| $\kappa$ | condition number of matrix $\boldsymbol{A}$, i.e. $\kappa = \lvert\lvert\lvert\boldsymbol{A}\rvert\rvert\rvert_2 \, \lvert\lvert\lvert\boldsymbol{A}^{-1}\rvert\rvert\rvert_2$ |
| $\mathrm{vec}(\boldsymbol{A})$ | vectorisation of $\boldsymbol{A}$ (i.e. stacking of columns) |
| $\mathrm{Tr}(\boldsymbol{A})$ | matrix trace $\mathrm{Tr}(\boldsymbol{A}) = \sum_i [\boldsymbol{A}]_{i,i}$ |
| $\Sigma$ | sum of matrix elements $\Sigma(\boldsymbol{A}) = \sum_{i,j} [\boldsymbol{A}]_{i,j}$ |

### Tensors

| | |
|---|---|
| $\mathbf{x}$ | a general tensor |
| $\mathbf{A}$ | a tensor operator of the form $\sum_i \boldsymbol{A}_i \otimes \boldsymbol{B}_i$ |
| $\mathsf{x}$ | a tensor (in canonical representation) |

| | |
|---|---|
| $(\boldsymbol{s}; \boldsymbol{X}, \boldsymbol{Y})$ | explicit canonical representation |

## Products

| | |
|---|---|
| $\langle \boldsymbol{x} \vert \boldsymbol{y} \rangle$ | inner product between vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ |
| $\langle \boldsymbol{x} \vert \boldsymbol{y} \rangle_{\boldsymbol{A}}$ | $\boldsymbol{A}$-inner product between vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ |
| $\langle \boldsymbol{A} \vert \boldsymbol{B} \rangle_F$ | Frobenius inner product between matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ |
| $f \circ g$ | Function composition, $(f \circ g)(x) = f(g(x))$ |
| $\boldsymbol{A} \odot \boldsymbol{B}$ | Hadamard product, $[\boldsymbol{A} \odot \boldsymbol{B}]_{i,j} = [\boldsymbol{A}]_{i,j}[\boldsymbol{B}]_{i,j}$ |
| $\boldsymbol{A} \otimes \boldsymbol{B}$ | Tensor product (see section 2) |
| $\boldsymbol{A} \hat{\otimes} \boldsymbol{B}$ | Kronecker product (see section 4.1.3) |
| $\boldsymbol{A} \check{\otimes} \boldsymbol{B}$ | Reversed Kronecker product, $\boldsymbol{A} \check{\otimes} \boldsymbol{B} = \boldsymbol{B} \hat{\otimes} \boldsymbol{A}$ |
| $\mathcal{U} \otimes \mathcal{V}$ | tensor product between vector spaces |
| $\mathcal{U} \times \mathcal{V}$ | Cartesian product between vector spaces |

## Norms

| | |
|---|---|
| $\| \cdot \|_p$ | Standard $L_p$ norm for functions or $p$ norm for vectors |
| $\|\boldsymbol{a}\|$ | arbitrary vector norm of vector $\boldsymbol{a}$ |
| $\|\boldsymbol{a}\|_p$ | vector $p$-norm of vector $\boldsymbol{a}$ |
| $\|\boldsymbol{A}\|$ | vector norm of matrix $\boldsymbol{A}$ as element of a vector space |
| $\|\boldsymbol{A}\|_p$ | vector $p$-norm of matrix $\boldsymbol{A}$, e.g. $\|\boldsymbol{A}\|_2 = \|\mathrm{vec}(\boldsymbol{A})\|_2 = \|\|\boldsymbol{A}\|\|_F$ |
| $\|\|\boldsymbol{A}\|\|$ | matrix norm of $\boldsymbol{A}$ induced by some vector norm $\|\cdot\|$ |
| $\|\|\boldsymbol{A}\|\|_p$ | matrix norm of $\boldsymbol{A}$ induced by the vector $p$-norm |
| $\|\|\boldsymbol{A}\|\|_F$ | Frobenius norm of matrix $\boldsymbol{A}$ (also called Hilbert-Schmidt norm) |
| $\|\|\boldsymbol{A}\|\|_{S,p}$ | Schatten $p$-norm of matrix $\boldsymbol{A}$, $\|\|\boldsymbol{A}\|\|_{S,p} = \|\boldsymbol{\sigma}\|_p = (\sum_i \sigma_i^p)^{1/p}$, where $\boldsymbol{\sigma}$ is the vector of singular values of $\boldsymbol{A}$, note further that $\|\|\boldsymbol{A}\|\|_{S,2} = \|\|\boldsymbol{A}\|\|_F$ and $\|\|\boldsymbol{A}\|\|_{S,\infty} = \|\|\boldsymbol{A}\|\|_2$ |

## Algorithms

| | |
|---|---|
| $\mathrm{O}(f(n))$ | an upper bound for memory or runtime of an algorithm |
| $\Theta(f(n))$ | a tight bound for memory or runtime of an algorithm |
| $(A, B, \ldots)$ | in algorithms: multiple return values |
| $a \leftarrow b$ | assignment: the value of expression $b$ is assigned to variable $a$ |

## Stochastics

| | |
|---|---|
| $\Omega$ | sample space |
| $\omega$ | elementary event, $\omega \in \Omega$ |
| $\mathcal{F}$ | sigma algebra on $\Omega$, $\mathcal{F} \subset 2^\Omega$ |
| $\mathcal{B}(A)$ | Borel sigma algebra, minimal sigma algebra that contains the open sets of $A$ (typically $A = \mathbb{R}^n$) |
| $\mathbb{P}$ | probability measure $\mathbb{P} : \mathcal{F} \to [0, 1]$ |
| $\sigma(X_1, X_2, \ldots)$ | sigma algebra generated by the random variables $X_1, X_2, \ldots$ |
| $\sigma(H)$ | sigma algebra generated by all random variables in the set $H$ |
| $(\Omega, \mathcal{F}, \mathbb{P})$ | probability space with sample space $\Omega$, sigma algebra $\mathcal{F}$ and probability measure $\mathbb{P}$ |
| $\mathbb{E}[X]$ | Expectation of random variable X, i.e. $\mathbb{E}[X] = \int_\Omega X(\omega) \mathrm{d}\mathbb{P}(\omega)$ |
| $\bar{X}$ | mean of random variable $X$, i.e. $\bar{X} = \mathbb{E}[X]$ |
| $\sigma_X^2$ | variance of $X$, i.e. $\sigma_X^2 = \mathbb{E}\left[(X - \bar{X})^2\right]$ |
| $\sigma_X$ | standard deviation of $X$, i.e. $\sigma_X = \sqrt{\mathbb{E}\left[(X - \bar{X})^2\right]}$ |
| $\mathcal{N}(\mu, \sigma^2)$ | normal (Gaussian) distribution with mean $\mu$ and variance $\sigma^2$ |
| $\ln \mathcal{N}(\mu, \sigma^2)$ | log-normal distribution with location parameter $\mu$ and scale parameter $\sigma$ |

| | |
|---|---|
| $\mathcal{U}(a,b)$ | uniform distribution on the interval $[a,b]$ |
| $\mathrm{Beta}(a,b)$ | beta distribution with parameters $a$ and $b$ |
| $r(x,\omega)$ | a random field |
| $\mathrm{cov}_r$ | covariance function of $r$, i.e. $\mathrm{cov}_r(x,y) = \mathbb{E}\left[(r(x,\cdot)-\bar{r}(x))(r(y,\cdot)-\bar{r}(y))\right]$ |

## Multiindex notation

| | |
|---|---|
| $\alpha,\beta,\ldots$ | a multiindex $\alpha=(\alpha_1,\alpha_2,\ldots)$ with $\alpha_i\in\mathbb{N}_0$ and $\alpha_i\neq 0$ for only finitely many $i$ |
| $\lvert\alpha\rvert$ | order of a multiindex $\lvert\alpha\rvert=\alpha_1+\alpha_2+\cdots<\infty$ |
| $\alpha!$ | factorial of a multiindex $\alpha!=\alpha_1!\,\alpha_2!\cdots$ |
| $x^\alpha$ | sequence $x$ raised to some multiindex power $x^\alpha=x_1^{\alpha_1}x_2^{\alpha_2}\cdots$ |
| $\partial_\alpha$ | partial derivative $\partial_\alpha f=\prod_i \partial^{\alpha_i} f/\partial x_i^{\alpha_i}$ |
| $\binom{n}{\alpha}$ | multinomial coefficient $\binom{n}{\alpha}=n!/\alpha!$, where $n=\lvert\alpha\rvert$ |
| $\mathcal{J}$ | the set of all multiindices $\mathcal{J}=(\mathbb{N}_0)_c^{\mathbb{N}}$, i.e. the set of sequences in $\mathbb{N}_0$ with finite support |
| $\mathcal{I}$ | an arbitrary finite subset of $\mathcal{J}$ |
| $\mathcal{I}_{m,p}$ | the subset of multiindices of maximal order $p$ and length $m$, i.e. $\mathcal{I}_{m,p}=\{\alpha\in\mathcal{J}\,\vert\,\lvert\alpha\rvert\leq p,\alpha_i=0\ \forall i>m\}$ |

## Miscellaneous

| | |
|---|---|
| $\mathbb{R}$ | the set of real numbers |
| $\mathbb{C}$ | the set of complex numbers |
| $\mathbb{N}$ | the set of natural numbers $\{1,2,\ldots\}$ |
| $\mathbb{N}_0$ | the set of natural numbers including zero $\{0,1,2,\ldots\}$ |
| $\ell_2(\mathbb{N})$ | the space of square integrable sequences |
| $\mathcal{U},\mathcal{V}$ | some general vector spaces or Banach spaces |
| $\mathcal{L}(\mathcal{U},\mathcal{V})$ | the space of continuous linear operators from $\mathcal{U}$ to $\mathcal{V}$ |

| | |
|---|---|
| $\mathcal{U}^*$ | the dual space of *continuous* linear functionals on $\mathcal{U}$ |
| $\mathcal{H}, \mathcal{K}$ | Hilbert spaces |
| $d$ | spatial dimension |
| $r$ | a general random field |
| $\mathbb{R}^d$ | the Euclidean vector space of dimension $d$ |
| $\mathcal{D}$ | the spatial domain, i.e. an open and connected subset of $\mathbb{R}^d$ |
| $\partial\mathcal{D}$ | the boundary $\overline{\mathcal{D}} \setminus \mathcal{D}$ of the domain $\mathcal{D}$ |
| $\Gamma_D$ | the part of $\partial\mathcal{D}$ where Dirichlet boundary conditions are specified |
| $\Gamma_N$ | the part of $\partial\mathcal{D}$ where Neumann boundary conditions are specified |
| $L_2(\mathcal{D})$ | the space of square integrable functions on $\mathcal{D}$ |
| $H^1(\mathcal{D})$ | the Sobolev space of functions on $\mathcal{D}$ with weak derivatives in $L_2(\mathcal{D})$ |
| $H^1_E(\mathcal{D})$ | subspace of $H^1(\mathcal{D})$ fulfilling homogeneous Dirichlet boundary conditions on $\Gamma_D$ |
| $d(\cdot,\cdot)$ | a metric |
| $\partial/\partial x$ | partial derivative by variable $x$ |
| $\arg\min_x f(x)$ | the value of $x$ (or set of values) that minimises $f(x)$ |
| $\mathrm{root}_x f(x)$ | the value of $x$ (or set of values) for which $f(x) = 0$ |
| $T_\varepsilon$ | truncation operator, i.e. $\|T_\varepsilon(x) - x\| \leq \varepsilon$ and or $\|T_\varepsilon(x) - x\| \leq \varepsilon\|x\|$ for relative truncations |
| $\Phi^{(k)}$ | iteration operator $x^{(k+1)} = \Phi^{(k)}(x^{(k)})$ |
| Id | identity operator $\mathrm{Id}(x) = x$ |

## A.2 Usage of fonts for tensor quantities

This work is mostly concerned with vectors and operators acting upon them. As vectors come in many disguises, being only defined by the abstract property of a vector space, it is sometimes necessary to be able to distinguish what kind of vector is meant visually. Therefore, different fonts are used in this thesis.

- For general vectors, e.g. elements of a Banach space or Hilbert space, the normal italic math font is used (Example: $u$).
- For vectors from the (Euclidean) vector space $\mathbb{R}^N$ a bold font is used (Example: $\boldsymbol{u}$).
- For general tensors as elements of an abstract tensor space, where the concrete space or representation is unspecified, an upright bold font is used (Example: $\mathbf{u}$).
- For tensors in canonical format a sans-serif font is used (Example: $\mathbf{u}$).

## A.3 A note on asymptotic notation

To describe asymptotic time and space complexity of algorithms, Landau notation is used in this thesis (see e.g. [18, 50]). In Landau notation, the fact that some function $f(n)$ grows asymptotically slower or equal to a function $g(n)$ as $n \to \infty$, i.e. there is a $C$ and a $N \in \mathbb{N}$ such that $f(n) \leq Cg(n)$ for all $n \geq N$, is expressed as

$$f(n) \in \mathrm{O}(g(n)). \tag{A.1}$$

For comparison of algorithms, however, one needs to be more specific than this. For example, for a function $f(n) = n^p$ it also holds that $f(n) \in \mathrm{O}(n^q)$ for any $q \geq p$. Thus, an algorithm with linear runtime complexity is $\mathrm{O}(n)$, but also $\mathrm{O}(n^2)$ or $\mathrm{O}(n^3)$. Thus, it would be a non sequitur to claim that an algorithm is "bad" if, for example, it has runtime complexity $\mathrm{O}(n^3)$ compared to one that has $\mathrm{O}(n^2)$.

What is needed for such comparisons is a tighter bound. This can be specified by the $\Theta$ notation, in which the symbol $\Theta$ means "exact order", i.e.

$$f(n) \in \Theta(g(n)), \tag{A.2}$$

if there are $C_1, C_2 > 0$ and $N \in \mathbb{N}$ such that $C_1 g(n) \leq f(n) \leq C_2 g(n)$ for all $n \geq N$. As this notation allows meaningful com-

parisons between algorithms and storage formats, it will be used throughout this thesis.

# Appendix B

# Computing tensor scalar products, norms and errors

Computing scalar products, norms and errors can often be defined in terms of each other and is therefore handled here together. The error between two elements of a Hilbert space is usually defined as the norm of their difference, and the norm is defined as the square root of the scalar product of a vector with itself. Thus all three problems can be reduced to computing the scalar product of two tensors, and will thus be the focus of this section. However, in cases for which this is not computationally optimal alternatives will be presented.

For computing the scalar product of two tensors we go back to the initial definition of the tensor product of two Hilbert spaces. Let $\mathbf{x} = \sum_i \boldsymbol{x}_i^{(1)} \otimes \boldsymbol{x}_i^{(2)}$ and $\mathbf{y} = \sum_i \boldsymbol{y}_i^{(1)} \otimes \boldsymbol{y}_i^{(2)}$, then

$$\langle \mathbf{x} | \mathbf{y} \rangle = \langle \sum_i \boldsymbol{x}_i^{(1)} \otimes \boldsymbol{x}_i^{(2)} | \sum_i \boldsymbol{y}_i^{(1)} \otimes \boldsymbol{y}_i^{(2)} \rangle \tag{B.1}$$

$$= \sum_{i,j} \langle \boldsymbol{x}_i^{(1)} | \boldsymbol{y}_j^{(1)} \rangle \langle \boldsymbol{x}_i^{(2)} | \boldsymbol{y}_j^{(2)} \rangle \tag{B.2}$$

Using the notation $\mathbf{x} = (\boldsymbol{X}^{(1)}, \boldsymbol{X}^{(2)})$ this can be written (and efficiently evaluated) as

$$\langle \mathbf{x} | \mathbf{y} \rangle = \Sigma((\boldsymbol{X}^{(1)T} \boldsymbol{Y}^{(1)}) \odot (\boldsymbol{X}^{(2)T} \boldsymbol{Y}^{(2)})) \tag{B.3}$$

where $\odot$ is the Hadamard product and the sum-function $\Sigma(\cdot)$ maps its argument to the sum its elements. If $\mathbf{x}$ and $\mathbf{y}$ are tensors

of size $N^{(1)} \times N^{(2)}$ and of rank $R$ and $S$ respectively, then this operation takes the runtime complexity of $\Theta((N^{(1)} + N^{(2)})RS)$ elementary operations (i.e. additions or multiplications) and the space complexity of $\Theta(RS)$.

If the scalar product would be computed by first forming the full tensors, equivalent to evaluating

$$\langle \mathbf{x}|\mathbf{y}\rangle = \Sigma((\boldsymbol{X}^{(1)}\boldsymbol{X}^{(2)T}) \odot (\boldsymbol{Y}^{(1)}\boldsymbol{Y}^{(2)T})) \tag{B.4}$$

the runtime complexity becomes $\Theta(N^{(1)}N^{(2)}(R+S))$ and space complexity $\Theta(N^{(1)}N^{(2)})$, thus being greatly inferior to the method above.

Tensor norms and errors can be easily computed from Eq. (B.3) by

$$\|\mathbf{x}\| = \langle \mathbf{x}|\mathbf{x}\rangle^{1/2} \tag{B.5}$$

and the difference $e_{\mathbf{x},\mathbf{y}}$ between to tensors by

$$e_{\mathbf{x},\mathbf{y}} = \|\mathbf{x} - \mathbf{y}\|. \tag{B.6}$$

However, note that in the case that $\mathbf{x} \approx \mathbf{y}$ Eq. (B.6) in conjunction with Eq. (B.3) suffers severely from cancellation. (For example, for small tensors of size $50 \times 50$ and ranks under 10 the formula above showed relative errors of $\approx 10^{-7}$ using double precision arithmetic).

A better approach to compute the norm in case the tensor is approximately zero and cancellation happens is the following: Compute the economy QR decompositions of $\boldsymbol{X}^{(1)}$ and $\boldsymbol{X}^{(2)}$, namely $\boldsymbol{X}^{(1)} = \boldsymbol{Q}^{(1)}\boldsymbol{R}^{(1)}$ and $\boldsymbol{X}^{(2)} = \boldsymbol{Q}^{(2)}\boldsymbol{R}^{(2)}$. Then

$$\|\mathbf{x}\|_2 = \|\|\boldsymbol{Q}^{(1)}\boldsymbol{R}^{(1)}\boldsymbol{R}^{(2)T}\boldsymbol{Q}^{(2)T}\|\|_F = \|\|\boldsymbol{R}^{(1)}\boldsymbol{R}^{(2)T}\|\|_F, \tag{B.7}$$

using the fact that the Frobenius norm is unitarily invariant. Complexity of this algorithm is $\Theta((N^{(1)} + N^{(2)})R^2 + N^{(1)}N^{(2)})$ and thus comparable with the complexity of Eq. (B.3).

When translating this approach to the case of inner products between tensors $\mathbf{x}$ and $\mathbf{y}$ it is important that the orthogonalization be made with respect to the *same bases*. So, let this time

$\begin{bmatrix} \boldsymbol{X}^{(1)} & \boldsymbol{Y}^{(1)} \end{bmatrix} = \boldsymbol{Q}^{(1)} \begin{bmatrix} \boldsymbol{R}_X^{(1)} & \boldsymbol{R}_Y^{(1)} \end{bmatrix}$ and correspondingly for the second index, then Eq. (B.3) becomes

$$\langle \mathbf{x} | \mathbf{y} \rangle = \Sigma((\boldsymbol{R}_X^{(1)T} \boldsymbol{R}_Y^{(1)}) \odot (\boldsymbol{R}_X^{(2)T} \boldsymbol{R}_Y^{(2)})) \tag{B.8}$$

and Eq. (B.4) becomes

$$\langle \mathbf{x} | \mathbf{y} \rangle = \Sigma((\boldsymbol{R}_X^{(1)} \boldsymbol{R}_X^{(2)T}) \odot (\boldsymbol{R}_Y^{(1)} \boldsymbol{R}_Y^{(2)T})) \tag{B.9}$$

The complexity of the QR decompositions is $\Theta((R+S)^2(N^{(1)} + N^{(2)}))$ and the evaluation of Eq. (B.8) and Eq. (B.9) require $\Theta((R+S)RS)$ and $\Theta((R+S)^3)$ operations respectively (noting that $R_A^{(1,2)} \in \mathbb{R}^{(R+S)\times R}$ and $R_B^{(1,2)} \in \mathbb{R}^{(R+S)\times S}$), thus being asymptotically equivalent. However, the second formula does not suffer from cancellation as the first one, as demonstrated in the following example.

**Example B.1.** *A tensor* $\mathbf{x}$ *of size* $153 \times 147$ *and rank* 13 *was randomly generated such that* $\|\mathbf{x}\|_2 = 1$. *Then for several values of* $\delta$ *from* $10^{-18}$ *to* $10^{-4}$ *modified tensors* $\tilde{\mathbf{x}}$ *were generated such that the true error was* $\|\tilde{\mathbf{x}} - \mathbf{x}\|_2 = \delta$ *and compared to errors computed by the different methods (see fig. B.1). It can be seen that any method relying on the inner product formula Eq.* (B.3) *has inferior accuracy levelling off at* $10^{-8}$, *while both formulas based on full tensor products have maximum accuracy. Note that it makes no difference, whether the summation is carried out normally or more sophisticated summation algorithms that avoid cancellation (e.g. Shewchuk's summation algorithm [82]) are employed.*

**Remark B.2.** *The algorithms described above are implemented in* sglib *in the file* `tensor/tensor_scalar_product`, *in which the Boolean parameter* `orth` *determines whether the accurate Eq.* (B.9) *is used or the slower, but more accurate Eq.* (B.3).

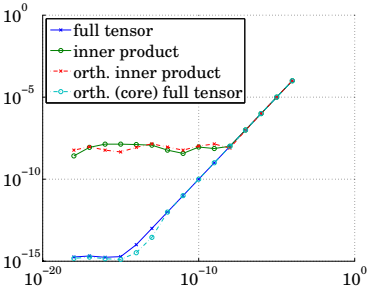



**Figure B.1:** Comparison of the computation of the difference between similar tensors $\mathbf{x}$ and $\tilde{\mathbf{x}}$ using different methods of computing the tensor inner product. On the horizontal axis is the true difference $\delta = \|\tilde{\mathbf{x}} - \mathbf{x}\|_2$ and on the vertical axis the computed difference. (Script: `tests/show_norm_errors`)

# List of Figures

# List of Tables

# List of Algorithms

# Bibliography


[1] H. Abdi, "Singular Value Decomposition (SVD) and Generalized Singular Value Decomposition", in *Encyclopedia of Measurement and Statistics*, N. J. Salkind, ed., SAGE Publications, Thousand Oaks (CA), 2007, 907–912, ISBN: 978-1412916110 (see p. 35).

[2] C. A. Andersson and R. Bro, "The N-way toolbox for MATLAB", *Chemometrics and Intelligent Laboratory Systems*, vol. 52, no. 1, 1–4, Aug. 2000, DOI: 10.1016/S0169-7439(00)00071-X (see p. 45).

[3] A. C. Antoulas, *Approximation of large-scale dynamical systems*. Cambridge University Press, Cambridge, 2005, ISBN: 978-0-89871-529-3 (see pp. 32, 56).

[4] K. Atkinson, *The Numerical Solution of Integral Equations of the Second Kind*. Cambridge University Press, Cambridge, 1997, ISBN: 978-0-52110-283-4 (see p. 32).

[5] I. Babuska, R. Tempone, and G. E. Zouraris, "Galerkin finite element approximations of stochastic elliptic partial differential equations", *SIAM Journal on Numerical Analysis*, vol. 42, no. 2, 800, 2004, DOI: 10.1137/S0036142902418680. URL: http://epubs.siam.org/doi/abs/10.1137/S0036142902418680 (see p. 3).

[6] B. W. Bader and T. G. Kolda, "Algorithm 862: MATLAB tensor classes for fast algorithm prototyping", *ACM Transactions on Mathematical Software*, vol. 32, no. 4, 635–653, Dec. 2006. DOI: 10.1145/1186785.1186794 (see p. 67).

[7] B. W. Bader and T. G. Kolda, *MATLAB Tensor Toolbox Version 2.3*, Jul. 2009. URL: http://csmr.ca.sandia.gov/~tgkolda/TensorToolbox/ (see pp. 45, 67).

[8] J. Ballani and L. Grasedyck, "A projection method to solve linear systems in tensor format", *Numerical Linear Algebra with Applications*, online, 2012, DOI: 10.1002/nla.1818. URL: http://onlinelibrary.wiley.com/doi/10.1002/nla.1818/abstract (see p. 6).

[9] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, 2nd ed. SIAM, Philadelphia, 1994 (see pp. 71, 72, 89–91, 97, 98).

[10] M. W. Berry, "Large scale sparse singular value computations", *International Journal of Supercomputer Applications*, vol. 6, 13–49, 1992. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.37.8591 (see p. 36).

[11] M. W. Berry, *SVDPACK*. URL: http://www.netlib.org/svdpack/ (see p. 36).

[12] M. Bieri, "Sparse tensor discretizations of elliptic PDEs with random input data", Dissertation, ETH Zürich, 2009. URL: http://e-collection.library.ethz.ch/eserv/eth:339/eth-339-02.pdf (see p. 7).

[13] M. Bieri, R. Andreev, and C. Schwab, "Sparse tensor discretization of elliptic sPDEs", *SIAM Journal on Scientific Computing*, vol. 31, no. 6, 4281–4304, 2010, DOI: 10.1137/090749256 (see p. 7).

[14] M. Brand, "Fast low-rank modifications of the thin singular value decomposition", *Linear Algebra and its Applications*, vol. 415, no. 1, 20–30, May 2006, DOI: 16/j.laa.2005.07.021 (see pp. 63, 65).

[15] S. C. Brenner and R. Scott, *The Mathematical Theory of Finite Element Methods*, 3rd ed. Springer, New York, 2007, ISBN: 0387759336 (see p. 18).

[16] I. N. Bronstein and K. A. Semendjajew, *Taschenbuch der Mathematik*, 16th ed. Verlag Harri Deutsch, Zürich, 1976, ISBN: 3-87144-016-7 (see p. 77).

[17] M. Chevreuil and A. Nouy, "Model order reduction based on proper generalized decomposition for the propagation of uncertainties in structural dynamics", *International Journal for Numerical Methods in Engineering*, vol. 89, no. 2, 241–268, 2012, DOI: 10.1002/nme.3249 (see p. 5).

[18] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Second Edition*, 2nd ed. The MIT Press, Cambridge, MA, Sep. 2001, ISBN: 0262032937 (see p. 148).

[19] J. Dennis, Jr. and R. Schnabel, *Numerical methods for unconstrained optimization and nonlinear equations*. SIAM, Philadelphia, PA, 1996, ISBN: 978-0898713640 (see p. 72).

[20] A. Doostan, R. G. Ghanem, and J. Red-Horse, "Stochastic model reduction for chaos representations", *Computer Methods in Applied Mechanics and Engineering*, vol. 196, no. 37-40, 3951–3966, Aug. 2007, DOI: 16/j.cma.2006.10.047 (see pp. 7, 32, 35).

[21] T. A. El Moselhy and Y. Marzouk, "An adaptive iterative method for high-dimensional stochastic PDEs", Preprint, 2011 (see pp. 7, 29).

[22] M. Espig, "Effiziente Bestapproximation mittels Summen von Elementartensoren in hohen Dimensionen", Dissertation, Universität Leipzig, 2008 (see p. 68).

[23] A. Falcó and A. Nouy, "A proper generalized decomposition for the solution of elliptic problems in abstract form by using a functional Eckart–Young approach", *Journal of Mathematical Analysis and Applications*, vol. 376, no. 2, 469–480, 2011, DOI: 10.1016/j.jmaa.2010.12.003 (see p. 6).

[24] A. Falcó and A. Nouy, "Proper generalized decomposition for nonlinear convex problems in tensor Banach spaces", *arXiv:1106.4424*, Jun. 2011. URL: http://arxiv.org/abs/1106.4424 (see p. 6).

[25] P. Frauenfelder, C. Schwab, and R. A. Todor, "Finite elements for elliptic problems with stochastic coefficients", *Computer Methods in Applied Mechanics and Engineering*, vol. 194, no. 2-5, 205–228, Feb. 2005, DOI: 10.1016/j.cma.2004.04.008 (see pp. 28, 32).

[26] Free Software Foundation, *The GNU General Public License version 3, 29 June 2007*. URL: http://www.gnu.org/copyleft/gpl.html (see p. 9).

[27] R. Ghanem and R. Kruger, "Numerical solutions of spectral stochastic finite element systems", *Computer Methods in Applied Mechanics and Engineering*, vol. 129, no. 3, 289–303, 1996 (see pp. 27, 104).

[28] R. Ghanem and P. Spanos, *Stochastic finite elements—A spectral approach*. Springer, New York, 1991, ISBN: 978-0387974569 (see pp. 3, 17, 26, 27, 29, 32).

[29] L. Giraud and J. Langou, "A robust criterion for the modified Gram–Schmidt algorithm with selective reorthogonalization", *SIAM Journal on Scientific Computing*, vol. 25, no. 2, 417, 2003, DOI: 10.1137/S106482750340783X (see p. 56).

[30] L. Giraud, J. Langou, and M. Rozloznik, "The loss of orthogonality in the Gram-Schmidt orthogonalization process", *Computers & Mathematics with Applications*, vol. 50, 1069–1075, Oct. 2005, ACM ID: 1665582, DOI: 10.1016/j.camwa.2005.08.009 (see pp. 56, 64).

[31] M. S. Gockenbach, *Understanding and implementing the finite element method*. SIAM, 2006, ISBN: 978-0-89871-614-6 (see p. 18).

[32] G. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed. John Hopkins University Press, Baltimore, MD, 1996, ISBN: 978-0801854149 (see pp. 32, 89).

[33] L. Grasedyck, "Hierarchical singular value decomposition of tensors", *SIAM Journal on Matrix Analysis and Applications*, vol. 31, no. 4, 2029, 2010, DOI: 10.1137/090764189 (see p. 68).

[34] L. Grasedyck and W. Hackbusch, "Construction and arithmetics of $\mathcal{H}$-matrices", *Computing*, vol. 70, no. 4, 295–334, 2003. URL: http://portal.acm.org/citation.cfm?id=958738 (see pp. 59–61).

[35] A. Greenbaum, "Behavior of slightly perturbed Lanczos and conjugate-gradient recurrences", *Linear Algebra and its Applications*, vol. 113, 7–63, Feb. 1989, DOI: 16/0024-3795(89)90285-1 (see p. 75).

[36] A. Greenbaum, *Iterative methods for solving linear systems*. SIAM, Philadelphia, 1997, ISBN: 978-0-89871-396-1 (see pp. 19, 71, 89, 90, 97, 99, 101).

[37] W. Hackbusch and S. Kühn, "A new scheme for the tensor representation", *Journal of Fourier Analysis and Applications*, vol. 15, no. 5, 706–722, Oct. 2009, DOI: 10.1007/s00041-009-9094-9 (see p. 6).

[38] W. Hackbusch, B. N. Khoromskij, and E. E. Tyrtyshnikov, "Approximate iterations for structured matrices", *Numerische Mathematik*, vol. 109, no. 3, 365–383, 2008, DOI: 10.1007/s00211-008-0143-0 (see p. 74).

[39] J. C. Hamano, S. O. Pearce, and L. Torvalds, *Git – version control system*. URL: http://git-scm.com/ (see p. 9).

[40] S. Hammarling and C. Lucas, "Updating the QR factorization and the least squares problem", MIMS Preprint, Nov. 2008. URL: http://eprints.ma.man.ac.uk/1192/ (see p. 65).

[41] R. A. Horn and C. R. Johnson, *Topics in matrix analysis*. Cambridge University Press, Cambridge, Jun. 1994, ISBN: 978-0-521-46713-1 (see p. 51).

[42] S. Janson, *Gaussian Hilbert Spaces*. Cambridge University Press, Cambridge, Jul. 1997, ISBN: 0521561280 (see p. 26).

[43] I. Jolliffe, *Principal Component Analysis*, Springer Series in Statistics. Springer, Berlin, 2002, ISBN: 978-0-387-95442-4 (see p. 35).

[44] K. Karhunen, "Über lineare Methoden in der Wahrscheinlichkeitsrechnung", *American Academic Science*, vol. 37, 3–79, 1947 (see p. 29).

[45] A. Keese, "Numerical solution of systems with stochastic uncertainties: A general purpose framework for stochastic finite elements", Dissertation, Technische Universität Braunschweig, Braunschweig, 2004. URL: http://www.digibib.tu-bs.de/?docid=00001595 (see pp. 17, 20, 26, 31, 32, 38, 42).

[46] C. T. Kelley, *Iterative methods for linear and nonlinear equations*. SIAM, Philadelphia, 1995, ISBN: 978-0-89871-352-7 (see pp. 19, 89).

[47] B. N. Khoromskij, A. Litvinenko, and H. G. Matthies, "Application of hierarchical matrices for computing the Karhunen–Loève expansion", *Computing*, vol. 84, no. 1-2, 49–67, Oct. 2008, DOI: 10.1007/s00607-008-0018-3 (see p. 32).

[48] B. N. Khoromskij, "Tensors-structured numerical methods in scientific computing: Survey on recent advances", *Chemometrics and Intelligent Laboratory Systems*, vol. 110, no. 1, 1–19, 2011. DOI: 10.1016/j.chemolab.2011.09.001 (see p. 45).

[49] B. N. Khoromskij and C. Schwab, "Tensor-structured Galerkin approximation of parametric and stochastic elliptic PDEs", *SIAM Journal on Scientific Computing*, vol. 33, no. 1, 364, 2011, DOI: 10.1137/100785715 (see p. 6).

[50] D. E. Knuth, "Two notes on notation", *Amer. Math. Monthly*, vol. 99, no. 5, 403–422, Apr. 1992. URL: http://arxiv.org/abs/math/9205211 (see p. 148).

[51] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications", *SIAM Review*, vol. 51, no. 3, 455–500, Sep. 2009. DOI: 10.1137/07070111X (see pp. 46, 54, 66, 67).

[52] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications", Sandia National Laboratories, Albuquerque, NM and Livermore, CA, Tech. Rep. SAND2007-6702, Nov. 2007. URL: http://www.prod.sandia.gov/cgi-bin/techlib/access-control.pl/2007/076702.pdf (see pp. 45, 46).

[53] P. Krée and C. Soize, *Mathematics of Random Phenomena—Random vibrations of mechanical structures*. D. Reidel, Dordrecht, 1986, ISBN: 978-9027723550 (see p. 33).

[54] M. Krosche and R. Niekamp, "Low rank approximation in spectral stochastic finite element method with solution space adaption", TU Braunschweig, Inst. f. Wissenschaftliches Rechnen, Informatikbericht 2010-03, 2010. URL: http://www.digibib.tu-bs.de/?docid=00036351 (see p. 5).

[55] L. D. Lathauwer, B. D. Moor, and J. Vandewalle, "A multilinear singular value decomposition", *SIAM J. Matrix Anal. Appl*, vol. 21, 1253–1278, 2000. DOI: 10.1137/S0895479896305696 (see pp. 6, 68).

[56] O. P. Le Maître and O. M. Knio, *Spectral Methods for Uncertainty Quantification: With Applications to Computational Fluid Dynamics*, 1st ed. Springer, New York, Apr. 2010, ISBN: 9048135192 (see pp. 17, 26).

[57] R. B. Lehoucq, D. C. Sorensen, and C. Yang, *Arpack User's Guide: Solution of Large-Scale Eigenvalue Problems With Implicityly Restarted Arnoldi Methods*. SIAM, Philadelpia, Apr. 1998, ISBN: 0898714079 (see p. 36).

[58] M. Loève, "Fonctions aletoires du second ordre", in *Processus stochastic et movement brownien*, P. Levy, ed., Gauthier Villars, Paris, 1948 (see p. 29).

[59] P. Malliavin, *Stochastic Analysis*. Springer, Berlin, 1997, ISBN: 978-3540570240 (see p. 42).

[60] MATLAB, *version 7.10.0 (R2010a)*. The MathWorks Inc., Natick, Massachusetts, 2010. URL: http://www.mathworks.com/help/techdoc/ (see pp. 36, 100, 124).

[61] H. G. Matthies, "Uncertainty quantification with stochastic finite elements", in *Encyclopedia of Computational Mechanics*, E. Stein, R. de Borst, and T. R. J. Hughes, eds., vol. 1 Fundamentals, John Wiley & Sons, Chichester, 2007, ch. 27, ISBN: 0-470-84699-2. DOI: 10.1002/0470091355.ecm071 (see pp. 3, 17, 19, 20, 29).

[62] H. G. Matthies and A. Keese, "Galerkin methods for linear and nonlinear elliptic stochastic partial differential equations", *Computer Methods in Applied Mechanics and Engineering*, vol. 194, no. 12-16, 1295–1331, Apr. 2005, DOI: 10.1016/j.cma.2004.05.027 (see pp. 3, 20, 24, 27).

[63] H. G. Matthies and E. Zander, "Solving stochastic systems with low-rank tensor compression", *Linear Algebra and its Applications*, vol. 436, no. 10, 3819–3838, 2012, DOI: 10.1016/j.laa.2011.04.017 (see pp. 75, 76).

[64] H. G. Matthies and E. Zander, "Sparse representations in stochastic mechanics", in *Computational Methods in Stochastic Dynamics*, M. Papadrakakis, G. Stefanou, and V. Papadopoulos, eds., vol. 22, Computational Methods in Applied Sciences, Springer Netherlands, 2011, 247–265, ISBN: 978-90-481-9987-7. DOI: 10.1007/978-90-481-9987-7_13 (see p. 75).

[65] F. Nobile, R. Tempone, and C. G. Webster, "An anisotropic sparse grid stochastic collocation method for partial differential equations with random input data", *SIAM Journal on Numerical Analysis*, vol. 46, no. 5, 2411, 2008, DOI: 10.1137/070680540 (see p. 28).

[66] A. Nouy, "A generalized spectral decomposition technique to solve a class of linear stochastic partial differential equations", *Computer Methods in Applied Mechanics and Engineering*, vol. 196, no. 45–48, 4521–4537, 2007, DOI: 10.1016/j.cma.2007.05.016 (see p. 5).

[67] A. Nouy, "A priori model reduction through proper generalized decomposition for solving time-dependent partial differential equations", *Computer Methods in Applied Mechanics and Engineering*, vol. 199, no. 23-24, 1603–1626, Apr. 2010, DOI: 10.1016/j.cma.2010.01.009 (see p. 5).

[68] A. Nouy, "Generalized spectral decomposition method for solving stochastic finite element equations: invariant subspace problem and dedicated algorithms", *Computer Methods in Applied Mechanics and Engineering*, vol. 197, no. 51–52, 4718–4736, 2008. DOI: 10.1016/j.cma.2008.06.012 (see p. 5).

[69] A. Nouy, "Proper generalized decompositions and separated representations for the numerical solution of high dimensional stochastic problems", *Archives of Computational Methods in Engineering*, Oct. 2010, DOI: 10.1007/s11831-010-9054-1 (see p. 5).

[70] W. Nowak, "Geostatistical methods for the identification of flow and transport parameters in the subsurface", Dissertation, Universität Stuttgart, 2005, ISBN: 3-9337 61-37-9. URL: http://elib.uni-stuttgart.de/opus/volltexte/2005/2275/ (see p. 31).

[71] J. Ortega and W. Rheinboldt, *Iterative solution of nonlinear equations in several variables*. SIAM, Philadelphia, PA, 2000, ISBN: 978-0898714616 (see pp. 72–74).

[72] I. V. Oseledets, "Tensor-Train decomposition", *SIAM Journal on Scientific Computing*, vol. 33, no. 5, 2295–2317, 2011, DOI: 10.1137/090752286 (see p. 68).

[73] I. V. Oseledets and E. E. Tyrtyshnikov, "Breaking the curse of dimensionality, or how to use SVD in many dimensions", *SIAM Journal on Scientific Computing*, vol. 31, no. 5, 3744–3759, 2009, DOI: 10.1137/090748330 (see p. 68).

[74] M. Pellissetti and R. Ghanem, "Iterative solution of systems of linear equations arising in the context of stochastic finite elements", *Advances in Engineering Software*, vol. 31, no. 8-9, 607–616, 2000 (see pp. 21, 104).

[75] K. B. Petersen and M. S. Pedersen, "The Matrix Cookbook", Technical University of Denmark, Tech. Rep., 2008, 67. URL: http://www.imm.dtu.dk/pubdb/views/edoc_download.php/3274/pdf/imm3274.pdf (see pp. 106, 107).

[76] K. K. Phoon, H. W. Huang, and S. T. Quek, "Simulation of strongly non-Gaussian processes using Karhunen-Loève expansion", *Probabilistic Engineering Mechanics*, vol. 20, no. 2, 188–198, 2005. DOI: 10.1016/j.probengmech.2005.05.007 (see p. 41).

[77] C. E. Powell and E. Ullmann, *Preconditioning stochastic Galerkin saddle point systems*, MIMS Preprint 2009.88, Nov. 2009. URL: http://eprints.ma.man.ac.uk/1354/ (see pp. 104–106).

[78] S. Roman, *Advanced Linear Algebra (Graduate Texts in Mathematics)*, 3rd ed., Graduate Texts in Mathematics. Springer, New York, Oct. 1995, ISBN: 0-387-97837-2 (see pp. 11, 14, 16).

[79] S. Roman, *The Umbral Calculus*. Dover Publications, Mineola, NY, Feb. 2005, ISBN: 0486441393 (see p. 43).

[80] R. A. Ryan, *Introduction to tensor products of Banach spaces*, Springer Monographs in Mathematics. Springer, London, Berlin, Heidelberg, Mar. 2002, ISBN: 1-85233-437-1 (see p. 11).

[81] S. Sakamoto and R. Ghanem, "Simulation of multi-dimensional non-Gaussian non-stationary random fields", *Probabilistic Engineering Mechanics*, vol. 17, no. 2, 167–176, 2002 (see pp. 38, 39).

[82] J. R. Shewchuk, "Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates", *Discrete & Computational Geometry*, vol. 18, no. 3, 305–363, Oct. 1997 (see p. 153).

[83] A. Sluis and H. A. Vorst, "The rate of convergence of conjugate gradients", *Numerische Mathematik*, vol. 48, no. 5, 543–560, 1986, DOI: 10.1007/BF01389450 (see p. 74).

[84] G. W. Stewart, "On the early history of the singular value decomposition", *SIAM Review*, vol. 35, no. 4, 551–566, Dec. 1993, URL: http://www.jstor.org/stable/2132388 (see p. 32).

[85] E. Ullmann, "A Kronecker product preconditioner for stochastic Galerkin finite element discretizations", *SIAM Journal on Scientific Computing*, vol. 32, no. 2, 923–946, Jan. 2010. DOI: 10.1137/080742853 (see pp. 104–106).

[86] E. Ullmann, "Solution strategies for stochastic finite element discretizations", Dissertation, Bergakademie Freiberg, 2008. URL: http://nbn-resolving.de/urn:nbn:de:bsz:105-8042820 (see pp. 27, 28).

[87] C. F. Van Loan, "The ubiquitous Kronecker product", *Journal of Computational and Applied Mathematics*, vol. 123, no. 1-2, 85–100, Nov. 2000, DOI: 16/S0377-0427(00)00393-9 (see pp. 51, 53).

[88] C. F. Van Loan and N. Pitsianis, "Approximation with Kronecker products", *Linear algebra for large scale and real time applications*, vol. 232, 293–314, 1993. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.42.1924 (see pp. 105, 106).

[89] H. A. van der Vorst, *Iterative Krylov methods for large linear systems*. Cambridge University Press, Cambridge, 2003, ISBN: 978-0-521-81828-5 (see pp. 19, 69, 71, 72, 97–99, 101, 109).

[90] N. Wiener, "The homogeneous chaos", *American Journal of Mathematics*, vol. 60, 897–936, 1938 (see p. 26).

[91] D. Xiu, *Numerical Methods for Stochastic Computations: A Spectral Method Approach*. Princeton University Press, Princeton, NJ, Jul. 2010, ISBN: 0691142122 (see p. 17).

[92] O. Zienkiewicz and R. Taylor, *The Finite Element Method—Volume 1, the basis*, 5th ed. Butterwort-Heinemann, Oxford, 2000 (see p. 18).